



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

UNIVERSITY OF ALBERTA

Query Optimization in Multidatabase Systems

BY

Ana M. Domínguez



A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

DEPARTMENT OF COMPUTING SCIENCE

Edmonton, Alberta
Fall 1993



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-88405-3

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Ana M. Domínguez

TITLE OF THESIS: Query Optimization in Multidatabase Systems

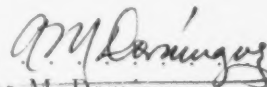
DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1993

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed) . . .



Ana M. Domínguez

102, 12215 Lansdowne Dr.

Edmonton, Alberta

T6H 4L4

Date:

October 8, 1993

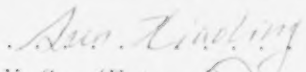
UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

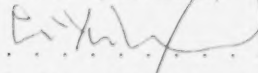
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Query Optimization in Multidatabase Systems** submitted by Ana M. Domínguez in partial fulfillment of the requirements for the degree of Master of Science.



Dr. M. T. Özsu (Supervisor)



Dr. X. Sun (External)



Dr. L. Yuan (Examiner)

Dr. P. Gburzynski (Chair)

Date: *October 7, 1993*

A papá Beto y mamá Tere

Abstract

Multidatabase systems provide interoperability among autonomous and potentially heterogeneous database systems. One of the problems that needs to be solved to provide true interoperability is the ability to efficiently process global queries that access multiple databases. Most of the approaches dealing with this problem have not taken into consideration the specific features of multidatabase systems, such as heterogeneity and autonomy, that make them different from distributed database systems. In this thesis, a solution that addresses heterogeneity and autonomy is presented. Our approach, which takes into account the *global* and *local* features present in a multidatabase system, follows algebraic optimization. To this end, an extended relational algebra is defined to incorporate the *move* operation which consists of moving data from one *component database* to another. The optimization methodology uses a cost model that embeds the local cost of processing into the global cost functions, defined for *total time cost* as well as for *response time*. The effectiveness of the methodology is tested by a performance study using a set of real query examples from a geo-information system application.

Acknowledgements

First of all, I want to express my infinite gratitude to my supervisor Dr. Tamer Özsu, not only for his wise guidance of my thesis, but also for providing us with a friendly and understanding environment to work. My very special thanks also to Dr. Patrick Valduriez for his always prompt answers while he was here at University of Alberta and through electronic mail. Besides, I want to thank the members of my committee Dr. Xiaoling Sun and Dr. Li-Yan Yuan.

Thanks to John Shillington and Delphine, from the LRIS for their unvaluable help with the geoinformation system presented for the experimentation.

I want to thank my brothers and sisters for their support not only now but all along my life. Very special thanks to my sisters Olga and María and my brother-in-law Saaka for just *being with me whenever I needed you*.

Thanks to all my friends at the department of Computing Science for making my stay here cheerful. No need for names, you know who you are.

And last but not at all least, I would like to thank my husband, Kofi, for his support, care, encouragement and understanding. You are the most special person in my life.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Scope	5
1.3	Thesis Organization	6
2	Related Work	7
2.1	Mermaid	8
2.2	MULTIBASE	11
2.3	Pegasus	13
3	Multidatabase Architecture	16
3.1	MDBS Query Processor	17
3.2	MDB Catalog	18
3.3	Component DBMSs	20
3.4	Component Interface Processors (CIP)	21
4	Query Processing Methodology	23
4.1	Query Processing Functional Layers	23

4.2	MDB Query Language (MQL)	26
4.3	Example	28
4.4	Query Processing Phases	32
4.4.1	Compilation Phase	34
4.4.2	Execution Phase	35
5	Optimization of Multidatabase Queries	38
5.1	Search Space	39
5.2	Transformation Rules	42
5.3	Search Strategy	46
5.4	Cost Functions	49
5.4.1	Response Time	49
5.4.2	Total Time	51
5.4.3	Component Cost Determination	52
6	Experimentation and Results	58
6.1	An Actual Multidatabase System	58
6.2	Implementation	60
6.3	Experimental Setup	62
6.3.1	Metrics	62
6.3.2	Instrumentation	63
6.4	Experiments and Results	70
6.4.1	Effect of Rules	70
6.4.2	Effect of Autonomy	72
6.4.3	Effect of Relation Sizes	75

6.4.4	Effect of Number of Components	78
6.4.5	Comparison of Initial and Optimized Cost	82
7	Conclusions and Future Work	84
7.1	Conclusions	84
7.2	Directions for Future Research	85
	Bibliography	87
A	Table Descriptions and Attributes	93

List of Figures

3.1	Functional Architecture of the Multidatabase System	17
4.1	Multidatabase Query Processing Layers	24
4.2	Algebraic Trees for Case 1	30
4.3	Algebraic Trees for Case 2	33
4.4	Compilation Phase	36
4.5	Execution Phase	37
5.1	Types of Query Processing Trees	40
5.2	Enumerative Search Algorithm-Using Heuristics	47
6.1	Multidatabase	60
6.2	Attributes of the Multidatabase Catalog Tables	68
6.3	Table Cardinalities	69
6.4	Different Sets of Rules by Priority (Response Time Cost) . . .	71
6.5	Data for Different Sets of Rules (Total Cost)	71
6.6	Effect of Autonomy (Response Time Cost)	73
6.7	Effect of Autonomy (Total Cost)	74
6.8	Effect of Relation Sizes (Response Time Cost)	76

6.9	Effect of Relation Sizes (Total Cost)	76
6.10	Delta Values for Response Time Cost	77
6.11	Delta Values for Total Time Cost	77
6.12	Effect of Number of Components (Response Time Cost)	79
6.13	Effect of Number of Components (Total Cost)	79
6.14	Effect of Number of Components (Response Time Cost)	80
6.15	Effect of Number of Components (Total Cost)	81
6.16	Initial Tree Cost, Optimized Tree Cost and Difference	83
A.1	ALTA Tables	95
A.2	LSAS Tables	96
A.3	LSDS Tables	97

Chapter 1

Introduction

1.1 Motivation

A *multidatabase system* (MDBS), as defined in [SL90], supports operations on multiple *component database systems* (CDBS), or as shown in [BGMS92], multidatabase systems allow users “access to data located in multiple autonomous database management systems.”

There are three main characteristics according to which database management systems (DBMSs) can be classified [ÖV91]. These orthogonal dimensions are distribution, heterogeneity and autonomy. Distribution refers to whether data are stored at a single computer system (centralized) or across multiple systems that are interconnected through a communication medium. Multidatabases may be distributed. Heterogeneity is due to differences in hardware, software and communication systems. The components of a multidatabase may be heterogeneous or homogeneous. Autonomy refers to the

degree of control that a database has on choosing its own design, communicating with other databases and executing local operations without external interference. The CDBS of an MDBS have, in general, high degree of autonomy.

An MDBS provides periodic, transaction-based exchange of data among multiple database management systems. It also provides access to all the CDBSs that compose the MDBS [SL90]. Multidatabase systems are one component of interoperable information systems.

Different problems arise in a multidatabase system. They may be classified into two groups: development problems and operating problems. Instances of the first group are schema translation, access control, negotiation and schema integration. Instances of the second group are query formulation, query processing and optimization, and transaction management. Also, integrity constraint problems are present.

There are two issues that make the aforementioned problems more complex in a multidatabase environment: heterogeneity and autonomy. We are particularly interested in the effects of these issues on query processing/optimization.

- **Heterogeneity** may be due to many different technological differences [SL90], e.g. differences in DBMSs (data models, system level support) or semantic heterogeneity. At another level, there may be differences in operating systems and hardware/communication. We are interested in the problems introduced by the differences in component multidatabase management systems (DBMS) as shown below:

- The optimization approaches of each CDBMS may be different. Some of them may do cost-based optimization whereas others may not. Even in the case that all the CDBMSs do cost-based optimization, the cost functions of each CDBMS may be different.
- The capabilities of each component DBMS may be different. Some may support an SQL¹ interface, others may not. Some may be relational, others non-relational. Some of the relational CDBMSs may allow the creation of temporary relations, others may not.
- **Autonomy** has a number of aspects. The CDBMSs may decide which operations the MDBS are allowed to perform and in which order. Also, the CDBMSs are free to choose the methods that they prefer for executing queries, and it is up to CDBMSs to let the *multidatabase management system* (MDBMS)² know information about cost functions used and statistics. Therefore, such details may not be available to the MDBS and cannot be manipulated by the MDB query optimizer³.

In this regard, [DKS92] identifies *proprietary DBMSs*, *conforming DBMSs*, and *non-conforming DBMSs* as three alternative levels of autonomy among the CDBMSs. They are classified depending on their capabilities and their integration into the MDBS.

- *Proprietary DBMS*. A CDBMS is called a proprietary DBMS if it

¹SQL stands for Structured Query Language.

²The MDBMS is the database management system of the multidatabase. The elements of the multidatabase are shown in Chapter 3.

³Again, the lector is referred to Chapter 3 for the definition of these terms.

is from the same vendor as the MDBMS, so that full information about its behaviour is available to the MDBMS developer.

- *Conforming DBMS*. A CDBMS is called a conforming DBMS if it is not from the same vendor as the MDBS, the component cost functions are not divulged but some statistics are available to the MDBMS.
- *Non-Conforming DBMS*. A CDBMS is called a non-conforming DBMS if neither the database statistics nor the cost functions of the CDBMSs are available to the MDBMS.

Query processing and optimization problems in multidatabase systems have not received sufficient and appropriate attention. Although some work has been done, it has been mainly focused on modifying distributed DBMS optimization algorithms to find a solution to the optimization problem in multidatabases. The major projects in this area are discussed in more detail in the next chapter [CBTY89, CER87, DH84]. There are certainly some similarities between distributed database systems and multidatabase systems in terms of query execution. To mention one, a query has to be subdivided into a set of subqueries, each of which is optimized further by component DBMSs. However, there are also fundamental differences between them [LS92]. These differences are generally introduced by the heterogeneity and autonomy issues which are not present in the distributed DBMS case. For example, the sites of a distributed DBMS are homogeneous and are connected through the same kind of network. In a multidatabase environment that may be the case but access is also permitted from and to heterogeneous component databases. In

the autonomy part, the distributed DBMS controls the query processing and optimization. The distributed DBMS also has access and decision on which the methods for accessing the data are going to be, which operations should be executed first, and so on. That is not the case in MDBSs as mentioned before.

1.2 Scope

This thesis focuses on the development of a query optimization methodology for multidatabase systems which takes into account the aforementioned characteristics. Query optimization is the problem of finding the optimum execution plan for a query in terms of the time of execution. The key difference between query optimization in distributed and MDBMSs is the way in which each of them views global and local optimization. In the distributed case, local and global optimization are controlled by the distributed MDBMS whereas in the MDB case the MDBMS has control only over the global optimization but each CDBMS controls local optimization. This fact has led us to focus our solution on this problem, particularly in the development of the cost functions and the transformation rules.

Our work defines a cost-based optimization technique for global queries. The global cost function is an aggregate of the cost of performing relational algebra operators at component DBMSs. The CDBS processing costs, communication costs and component temporary storage costs are embedded in the global function. Total time cost as well as response time cost functions are defined.

Transformation rules that take into account the alternative ways of moving data among components are defined to navigate the search space. The effect of those rules is presented in the results as well as the measures of both types of cost in a multidatabase environment.

1.3 Thesis Organization

This thesis is organized as follows. In Chapter 2, a review of previous related research is presented. Chapter 3 contains the multidatabase architecture in which the query optimization is to take place. The query processing layers including the details of the optimization process are presented in Chapter 4. The search strategy and the transformation rules are also discussed in Chapter 4. The cost model used for the optimizer is defined in Chapter 5. In Chapter 6, experimentation and results are presented. Finally, Chapter 7 contains the conclusions and future work.

Chapter 2

Related Work

The existing approaches to multidatabase query processing and optimization may be divided into those for production environments and those for research purposes [TTC⁺90]. These systems are discussed in this chapter as examples. The first two systems that are going to be discussed in this chapter (*Mermaid* and *MULTIBASE*) are being developed to become commercial products, and they are, up to a point, very well developed. It must be mentioned also that these systems make use mainly of distributed algorithms for optimizing multidatabase queries. The third one to be discussed, *Pegasus*, is a research project being developed at HP labs.

There have been other projects in which multidatabase query optimization has been studied. For example, [RC87] discusses algorithms to maximize local processing while minimizing data transfer between component DBSs. This is mainly a research project and most of their research has focused on query formulation and translation, and schema integration [RC85, RC89].

Another production project is DATAPLEX [Chu90] which is being developed by General Motors Research Laboratories. The main objective is to generate a plan that reduces the amount of data moved among computers.

The aforementioned projects are the most closely related ones to our research. Many other production and research projects are being developed [BHP92], but most of them emphasize issues different from query processing-optimization.

2.1 Mermaid

Mermaid is a system that allows users access to multiple databases stored under relational DBMSs running on different machines. The component DBMSs include commercial relational DBMSs on workstations and a database machine with a minicomputer host [TTC⁺90]. Mermaid is not a full-fledged database management system; rather, it is a *front-end system* that locates and integrates data that are maintained by local DBMSs [TBD⁺87].

The data can be manipulated using a common language, either SQL or ARIEL, a user-friendly query language developed at Systems Development Corporation (now part of Unisys) [TBD⁺87].

Each specific user interface requires a translator to translate the user language into DIL (Distributed Intermediate Language) which is designed for ease of translation. Requests issued using DIL are processed by a constructing processor that produces and executes a query plan. Transforming processors are used to translate DIL requests into the component query languages. These transformers also interact with component DBMSs and send

data to other transforming processors for some algebraic operations that require interaction between components.

Mermaid makes use of the Data Dictionary/Directory (DD/D) in order to do translations and query planning [TBD⁺87]. The DD/D contains information about the databases, the users, the DBMSs, the host computers, and the network.

The Mermaid system makes use of ideas in SDD-1 [BGW⁺81] and distributed INGRES [ESW78] for developing its query optimization algorithms [CBTY89]. SDD-1 and distributed INGRES algorithms are homogeneous distributed database query optimization algorithms.

It is assumed that processing costs and communication cost will not be uniform [TBD⁺87]. This is due to the fact that it is assumed that the component DBMSs may have different operational characteristics and that the network will also be nonuniform.

The main objective of query optimization is to minimize response time since it is assumed that users of Mermaid will tend to use it interactively [TBD⁺87]. The optimization algorithm has been designed to give a quick answer, not necessarily the optimal solution. The generated plan is not compiled or saved because *ad hoc* queries are expected.

Mermaid assumes the existence of local optimizers that will choose which indices to use, how to execute a query and the order of joins if multiple joins are required [TBD⁺87].

An integrated algorithm is used to optimize the processing cost as well as the transmission cost [CBTY89]. There is distributed processing of aggregates [CBTY89].

The integrated algorithm is composed of the following steps: (1) execution of selection clauses, (2) choice of the fragmented relation and processing sites, (3) semijoin application, (4) data transmission, (5) parallel query processing, (6) result assembly, and (7) final processing.

Semantic information is used for efficient query processing, particularly, for solving the fragmentation problem [CBTY89]. The *fragment attribute*, i.e., the attribute by which a relation is fragmented may be found in the selection clause and this information is used to reduce transmission and processing cost.

The main difference between Mermaid and our approach is that Mermaid considers optimization of ad hoc queries whereas we consider optimization of production queries. We have observed, from the real geo-information system on which our experimentation was tried and from other studies [Zhu92], that it is very likely that production queries will be present in a MDB environment and that it is useful to be able to optimize them at compile time since they are going to be used over and over again.

Regarding calculation of cost, Mermaid and our approach optimize response time cost. However, for the calculation of the processing cost, Mermaid calculates the selectivity factors while we obtain them using statistics.

Mermaid makes use of semijoins in its optimization algorithm. This requires major processing and optimization at CDBs. Nothing is mentioned about execution autonomy of the components. Execution autonomy gives components the ability to execute local operations without interference from external operations by other components and to decide whether to let other operations be executed and their order. We consider execution autonomy as

one of the major differences between distributed DBMSs and MDBMSs.

2.2 MULTIBASE

MULTIBASE offers an integrated schema and a single query language, DAPLEX [Day83]. It does not provide the capability to update the data in the local databases or to synchronize read operations across several sites.

The system has been designed to accomplish three objectives: generality, compatibility and extensibility. It is general because it has not been designed for any specific application area. It is compatible because it has been designed to incorporate a wide range of data sources and it is extensible because it has been designed to minimize the cost of adding a new data source.

There is a schema architecture and a component architecture. In its component architecture, there are three major types of components: the Global Data Manager, one or more Local Database Interfaces and the Internal DBMS.

The global DAPLEX query is modified into a DAPLEX query over the local schemas [LR82]. The processes used for this task are the following: a Global Query Optimizer that produces a global plan, a Decomposer that decomposes the global plan into single-site DAPLEX queries, a Filter that reduces the decomposed queries by removing operations that are not supported by the corresponding component DBMSs, and a Monitor that controls the distributed executions of the subqueries.

MULTIBASE performs query optimization at two levels: global and local [LR82]. At the global level, the objectives are to minimize the amount of

data that is moved between sites and to maximize the potential for parallel processing. At the local level, the objective is to minimize the amount of time it takes to retrieve data from local DBMSs by taking advantage of local query languages, physical database organization and fast access plans.

Local optimization deals with minimizing the time to retrieve data from a local DBMS. The queries that are sent to each local site are subjected to local access path optimization.

Global optimization deals with fully utilizing the potential for parallel processing in a distributed system and minimizing communication costs by reducing the amount of data moved between sites, using the techniques used in SDD-1 [LR82].

Query modification, global query optimization and the execution of post-processing queries are performed at a *special global site*.

In the tactics for processing conjunctive generalization queries, transformations for reducing the volume of data moved between sites are shown. Their cost function is a weighted sum of data movement costs and local processing costs. The aforementioned tactic is not always applicable to conjunctive generalization queries that involve selection, projection, or join over aggregated attributes.

An approach in which four reduction tactics are applied to joins is presented in [Day83]. These tactics are used on each join in a join order whenever they are applicable and beneficial in terms of reducing the cost. The tactics are the following: 1) Distributing selection over generalization, 2) Distributing joins over generalization, 3) Semijoin reduction, and 4) Semiouter join reduction.

Tree queries and *strategies* for solving them are presented, where *strategies* is understood to be the different states (or possible solutions) in the search space. The search space is defined as well [DG82].

In MULTIBASE, optimization is performed at global and at local levels. It is considered that local processing cost may be calculated based on fast access plans, physical database organization and the type of local query language. However, we consider that the information necessary for such calculations may not be available from the components. Furthermore, we consider that the MDBMS does not know the physical database organization since the components have the right to keep that information for themselves. Therefore, the two approaches are different and ours takes into account component autonomy, an important characteristics present in MDB environments. Both approaches optimize response time cost. The cost functions defined by MULTIBASE consider communication and local processing cost. In addition, we also consider temporary storage cost.

MULTIBASE makes use of a *special global site* and we make use of a *control database* to do execution of post-processing queries and to handle the MDBMS operations.

2.3 Pegasus

One of the most recent projects in this area is the Pegasus project at HP Labs [DKS92].

It makes use of the object-oriented data modeling and programming capabilities [ASD⁺91]. Type and function abstractions are used to solve the

problem of mapping and integration.

There are three functional layers in the Pegasus architecture. The first layer is the *intelligent information access layer* that provides services such as information mining, browsers and schema exploration. The second layer is the *cooperative information management layer* that deals with schema integration, global query processing, local query translation, and transaction management. And the third layer is the *local data access layer* which manages schema mapping, local query and command translation, network communications, local system invocation, data conversion and routing.

There are six types of elements participating in the Pegasus architecture. The *executive* manages the interaction between Pegasus and its clients. The *optimizer* produces a more efficient alternative plan that is equivalent to the original plan. *Local translators* do the mapping between local schema and equivalent imported schemas used to translate a Pegasus subquery into the local database language. The *global interpreter* dispatches and synchronizes internal and external executables. The *schema and object managers* implement data definition operations, catalog management, object management and schema integration services. And *Pegasus agents* are processes that run in the same machine as local DPMSs and represent Pegasus at the local sites.

Global query optimization is performed in Pegasus using the statistical information that is kept in the global data dictionary. This is achieved using the *statistic information processor*. A *plan generator* decomposes and schedules the query into subqueries that are sent to the affected components for execution by the *dispatcher*. The *subquery execution monitor* keeps track of the status of subquery execution and provides feedback to the statistic

information collector.

The objective of the global query optimizer is to minimize the response time of global queries. The expected response time of the component databases is computed using statistics related to the data in those particular components [LS92].

The way to predict the performance of the component databases is through the use of a calibrating database. A calibrating database is an artificially generated database built by generating values for each attribute based on formulas defined for the access method defined for each column. This database is then queried, the coefficients of the cost functions are deducted and the functions can be then used for calculating the cost [DKS92]. More details about these calculations are given in Section 5.4 which deals with the cost functions.

Pegasus is an ongoing project and therefore little information is known about the strategy used for optimization. What is known is the objective of optimizing response time cost, which is also our purpose. In their work on local query optimization prediction, autonomy heterogeneity and distribution are considered, therefore it is very likely that these aspects are also going to be considered in their search strategy. However, the definition of the space and the search strategy have not yet been reported.

Statistical information is used by Pegasus to do global and local optimization. We make use of their local processing cost prediction method and embed it into our global cost functions. They have not defined the global cost functions explicitly.

Chapter 3

Multidatabase Architecture

The architecture that is assumed in this study is shown in Figure 3.1, which shows the different components and the communication among them. Their functionality is described below.

It is assumed that we are working with components which are *conforming* database systems as defined in Section 1.1. That means that some statistics are available to the global query optimizer but not the cost functions. The statistics are going to be used for calculating the coefficients of the cost functions so that the cost of executing queries at the CDB may be computed.

For ease of presentation, the architecture is shown with two CDBs. This is done for clarity but the system can perfectly have more than two component databases.

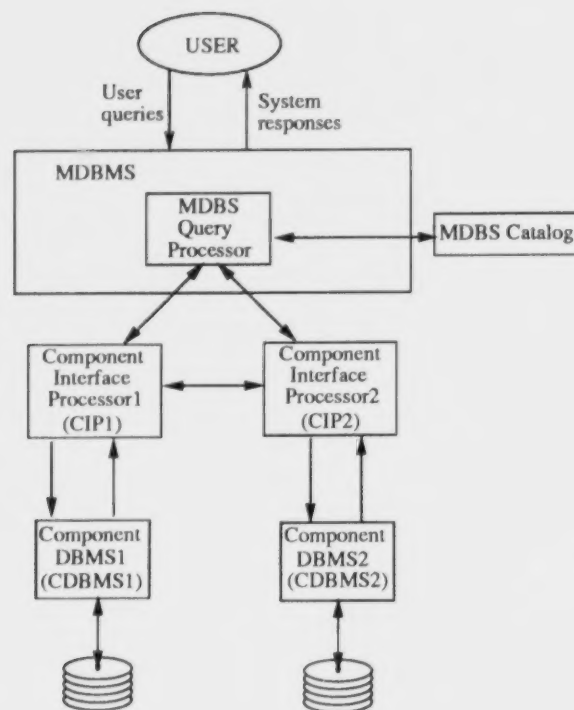


Figure 3.1: Functional Architecture of the Multidatabase System

3.1 MDBS Query Processor

The *MDBS query processor* is the receiver of the *global queries* posed to the multidatabase and it is in charge of optimizing it. The MDBS query processor may be implemented on top of any primary CDBMS. The MDBS query processor implements the full range of processing tasks including query analysis, query decomposition, query localization, query optimization and code generation. These steps are detailed in the chapter dealing with query processing (Chapter 4). The MDB catalog is used at this stage to get the

necessary processing information.

3.2 MDB Catalog

The MDB catalog contains information that is accessed by different processes of the query processing architecture in the MDBS. The MDB catalog has the following information in it: the global conceptual schema, the fragmentation schema and the cost function for each CDBMS. The cost function is used by the query optimizer to calculate the cost of queries posed to the CDBs. The information regarding the cost functions is statistical information and the coefficients of the cost function.

The *global conceptual schema* is a unified definition of the logical structure of the data at the CDBs that is available for global access¹. For instance, the definition of the relations, attributes, types and domain constraints are part of the global conceptual schema. The global conceptual schema is specified in relational model in this study. The translation of non-relational representations to a relational one is an activity that takes place during schema integration. This allows users to express all queries in a relational query language (SQL in this case).

The *fragmentation schema* contains the information about the horizontal partitioning of the relations, i.e. the definition of the fragments and their locations at the component databases. *Horizontal* fragmentation is the par-

¹Many researchers argue that in a multidatabase system there would not be a single global conceptual schema, but many - perhaps one per application domain. This complication is not necessary for this research and will not be introduced

tition of a relation in terms of tuples, whereas *vertical* fragmentation is the partition of a relation in terms of attributes [ÖV91]. Vertical fragmentation is not considered since it may be treated as the join of relations across component databases. Information about the location of any relation defined at the global level is also kept in the fragmentation schema.

The *cost model* (or cost function) for each CDB has the calibrated coefficients of the cost functions. The basic statistics that are needed for evaluating the cost functions are also kept as part of this information. The data related to the communication and storage capabilities of the components is stored here. This information is input to the module which calculates the component processing, communication and storage costs. The cost model for each component DBMS is specified in terms of the relational algebra operations, using a method similar to that of [DKS92]. We discuss the cost model in more detail in later sections.

The fragmentation schema information is used to perform query decomposition and localization of data. The cost model provides information to the query optimization stage. The global schema is used to do the conversions. The MDB Catalog is created and updated using the data definition language for the MDBS.

The content of the MDB catalog is illustrated with an example, which is also used and expanded in chapter 4.

At the *global conceptual schema* level, the following information is kept: There is a relation R which has three attributes: A, B, C. The attribute types are: A is numeric, with $1 \leq A \leq 200$, B and

C are strings.

At the *fragmentation schema* level, it is known that R is horizontally fragmented over CDB1 and CDB2 as follows:

$$R_1 = \sigma_{1 \leq A \leq 100} (R)$$

$$R_2 = \sigma_{100 < A \leq 200} (R)$$

R_1 contains those tuples of relation R whose A attributes have values between 1 and 100. R_2 is defined similarly.

The coefficients of the cost functions for CDB1 and CDB2 are found in the *cost model*. See the implementation part for more detailed information on what the coefficients are.

3.3 Component DBMSs

As stated earlier, an MDBS provides global access to the users over a number of possibly heterogeneous databases that may be distributed over a network of computers. These component databases may be catalogued as *primary components* or *secondary components*. *Primary components* are relational DBMSs and they provide an SQL interface whereas *secondary components* are non-relational. Some of the primary components are capable of creating and storing temporary relations. These primary components are especially important since they are the candidates for executing the operations that span over multiple components. For instance, as shown below, the MDBS layer of software is implemented on top of a primary DBMS with this capability. This component DBMS stores and manages the MDBS catalog. Thus,

there is always at least one primary component in the system where all query processing can be performed if necessary.

The CDBs are a source of information for the MDBS catalog, to keep the data concerned with the components up to date. That includes the fragment-relation definitions and statistical information, as well as the capabilities of the databases. There is no arrow in the diagram that shows this relationship between the components and the MDBS catalog since that would be maintenance activity of the system.

3.4 Component Interface Processors (CIP)

There is one *component interface processor* (CIP) for each component DBMS in the system. CIPs perform a number of functions. First, they control the execution of a given subquery by an associated component DBMS. They are responsible for translating the subquery to the particular data manipulation language of the component DBMS. Where data is moved between component DBMSs², CIPs are responsible for performing the format conversions and for creating the temporary relations. Also, the communication control among component DBMSs is done at this stage. When the execution schedule requires that the result of a component query be sent to another component DBMS for further processing, it is the CIPs that handle this data transfer. This may also be the case of the final result having to be sent to the requesting component DBMS. There are many other roles that CIPs play, particularly

²As we will discuss later, movement of data is from secondary components to primary ones which have temporary storage capabilities.

with respect to transaction processing. Those functions are not relevant for query processing/optimization, therefore they will not be discussed here.

CIPs are neither part of the MDBMS nor part of the CDBMSs. If each CDBMS is at a different site, CIPs are located at the sites of their corresponding CDBMSs. On the other hand, they are agents of the MDBMS. Therefore, they are associated with CDBMSs but perform functions requested by the MDBMS. In that sense, it is not appropriate to consider them either as part of the MDBMS or as parts of CDBMSs.

Chapter 4

Query Processing Methodology

Multidatabase query processing, similar to centralized and distributed query processing, is divided into different steps that have to be executed in order to get the answer to a given query posed to the MDB system. These steps are described in the following subsection. In order to perform these steps, an intermediate language becomes necessary. The need for such a language is explained in Section 4.2, which also includes the language definition.

4.1 Query Processing Functional Layers

The MDBS query processing methodology is presented in Figure 4.1. Some of the steps involved in this methodology are similar to those in distributed DBS query processing [ÖV91]. Others may need significant changes to accommo-

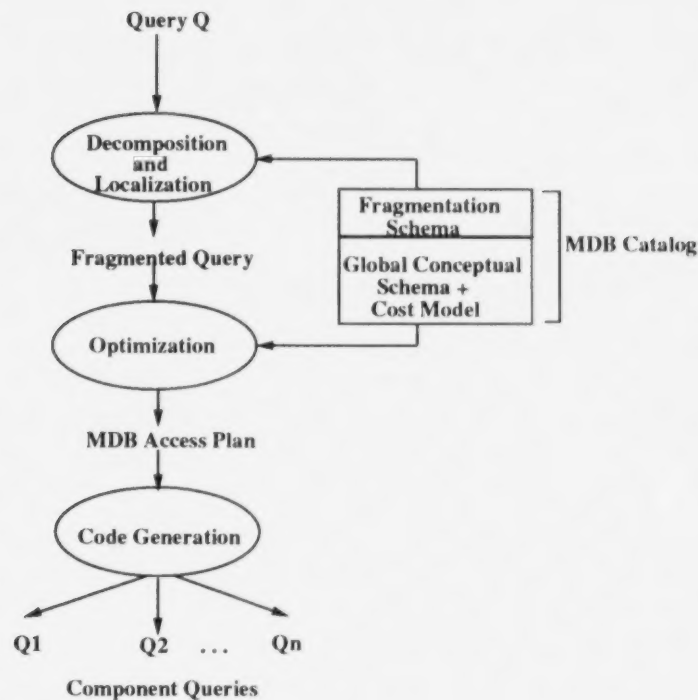


Figure 4.1: Multidatabase Query Processing Layers

date the characteristics of multidatabase query processing and optimization. Each step is mapped into the architecture presented in the previous chapter.

We assume that the user interface to the MDBS is SQL. Thus, the user submits a global query, expressed in SQL, which accesses multiple, autonomous databases, some of which may not be relational. The query is specified on a global schema, without any reference to the distribution of data among the autonomous databases or their fragmentation. During the *decomposition* and *localization* process, the query is analyzed for correctness and translated into an intermediate multidatabase algebra expression over

relations or fragments.

An extended relational algebra is used by the MDBS query processor called the *multidatabase query language* or MQL. It consists of the traditional relational algebra operators plus an operator to express the movement of data among CDBMSs, which specifies the transfer of the result of a subquery from one component DBMS to another one. We introduce the **move** operator for this purpose. It also specifies that temporary results are to be stored at the destination component DBMS (as temporary relations) to be further operated upon.

Extended algebraic expressions can be represented as processing trees (PT) [KBZ86]. As in the traditional algebra trees, the leaves of the PT are the relations or fragments stored in the component databases. The interior nodes in the tree are algebraic operators; the edges represent the intermediate results obtained from applying the corresponding algebra operator.

The *optimization* step takes a processing tree and generates an execution plan which specifies the subqueries and the transmission of intermediate results between component DBMSs. This process may take several steps, since transformation rules have to be applied to get equivalent processing trees from which the one with the lowest cost is chosen as the best execution plan for that particular query. Optimization is the major focus of this work and will be discussed further in the subsequent sections as well as in the chapter dealing with the implementation.

Finally, the *code generation* step takes the optimized processing tree and generates a set of subqueries (called *component queries*) to be executed by each component DBMS after being received from the corresponding CIP. At

this stage a **move** operation is separated into **send** and **receive** operations, each of which is included in one of the component queries. Code generation is the interface between the MDBS query processor/optimizer and the CIPs. CIPs take these component queries (specified in extended algebra), translate them into the interface language of each component DBMS, submit them to each component DBMS, control the transfer of data from one component DBMS to another, create and operate temporary tables based on incoming data from other component databases and destroy temporary tables after finalizing their use.

4.2 MDB Query Language (MQL)

As mentioned earlier, it is necessary to define a language in which the queries can be expressed such that localization and transmission information is available during optimization. There is a need to introduce the MQL because traditional relational algebra is not expressive enough to show and manipulate information generated by communication among components.

We have an extended relational algebra by introducing a **move** operator, which moves data among components. The **move** operator actually represents two operations, **send** and **receive**. The **send** operator takes an intermediate result generated by a CDBMS and sends it over to a specified CDBMS. The semantics of the **receive** operator is the opposite; it additionally creates and populates a temporary relation. The temporary relation created using the **receive** operator can be dropped at the end, after the query is executed. At the optimization level, the **move** operator is sufficient to express, in the

processing trees, the transmission of an intermediate result from one component to another. The **send** and **receive** operators are interpreted by the CIPs and are not visible to the CDBMSs, preserving their local autonomy. The remaining algebraic operations are the traditional relational algebra operations and rules, such as selection, join, projection, union, etc., as shown in [Ull82].

The following is the notation used to show operations and relations in the extended relational algebra and the semantics of the **move** operator. This notation is also used for the trees shown in the examples later.

- The MDB is composed of n CDBs.
- DB_i denotes a CDB where $1 \leq i \leq n$.
- R is relation defined on the MDB.
- R_i denotes the horizontal fragment of R at CDB_i , or in case R is not fragmented, that relation R is at component i .
- op_i denotes performing op at CDB_i .
- Q_{ij} denotes the j th component query executed at CDB_i .

Based on the notation described above, the semantics of the **move** operator are given as follows:

$move_j(R_i)$ represents sending relation R from CDB_i to CDB_j , where CDB_j is a primary component with temporary storage capabilities;

$send_j(R_i)$ sends relation R (it may be a partial result) from CDB_i to component j and creates a relation denoted R_{ij} ;

$receive_j(R_i)$ receives the relation R (it may be a partial result) at CDB_j from CDB_i and creates a relation denoted R_{ij} .

4.3 Example

We describe the query processing methodology by means of the following example:

Let us assume there are two relations, R and S . There are two CDBs, CDB_1 and CDB_2 . R has attributes A , B and C . S has attributes A , D and E . Consider the following SQL query Q :

```
SELECT A, B, E
FROM R, S
WHERE R.A = S.A
```

Let us assume the following fragmentation schema:

$$R = R_1 \cup R_2^1 \text{ and}$$

$$S = S_1$$

R is fragmented into R_1 , stored at CDB_1 , and into R_2 stored at CDB_2 . Relation S is not fragmented and is stored in CDB_1 . The join operation $\bowtie_A$ is the join of relations R and S on attribute A . Π is the projection of attribute A , and $\cup$ is the union of two relations. Numeric subscripts indicate the localization of a relation, fragment or operation.

Two cases can be distinguished depending on the models that the CDBs use. The first case to be treated is when CDB_1 has a relational model but CDB_2 does not. In the second case CDB_1 does not support a relational

¹This refers also to the example in section 3.2.

model but CDB_2 does. There is a third case, when both CDB_1 and CDB_2 support relational models. This case can be treated like case 1. It is assumed that the two CDBs are at the same machine, so that communication cost over a network is not needed to be considered.

The examples that are shown below present only the compilation phase.

Case 1

The first decomposition of query Q into the algebraic tree is shown in Figure 4.2.a. This step is executed by the MDBQP. This is step 1 in Figure 4.1, i.e., decomposition and localization process.

After applying distribution of the join operator over the union operator, the algebraic tree shown in Figure 4.2.b is obtained. Here the optimization process has started. This step is also performed by the MDBQP. Since CDB_1 is a relational DB and CDB_2 is not, it is preferable to process the query at CDB_1 . Therefore, R_2 is moved to CDB_1 (Figure 4.2.c).

The component queries are then formed by grouping as many component operations as possible, as shown in Figure 4.2.d. Two component queries are generated: Q_{11} and Q_{22} . The result of performing component query Q_{22} at CDB_2 is sent to CDB_1 where the answer to query Q is generated.

The code fragments created to get the result query Q as follows, where relation T_{ij} represents the j th temporary table created at CDB_i .

Code Fragment for Query Q_{11} . This is executed at $CDBS_1$.

```

SELECT A, B, E
FROM R1, S1
WHERE R1.A = S1.A
UNION
SELECT A, B, E
FROM T11, S1

```

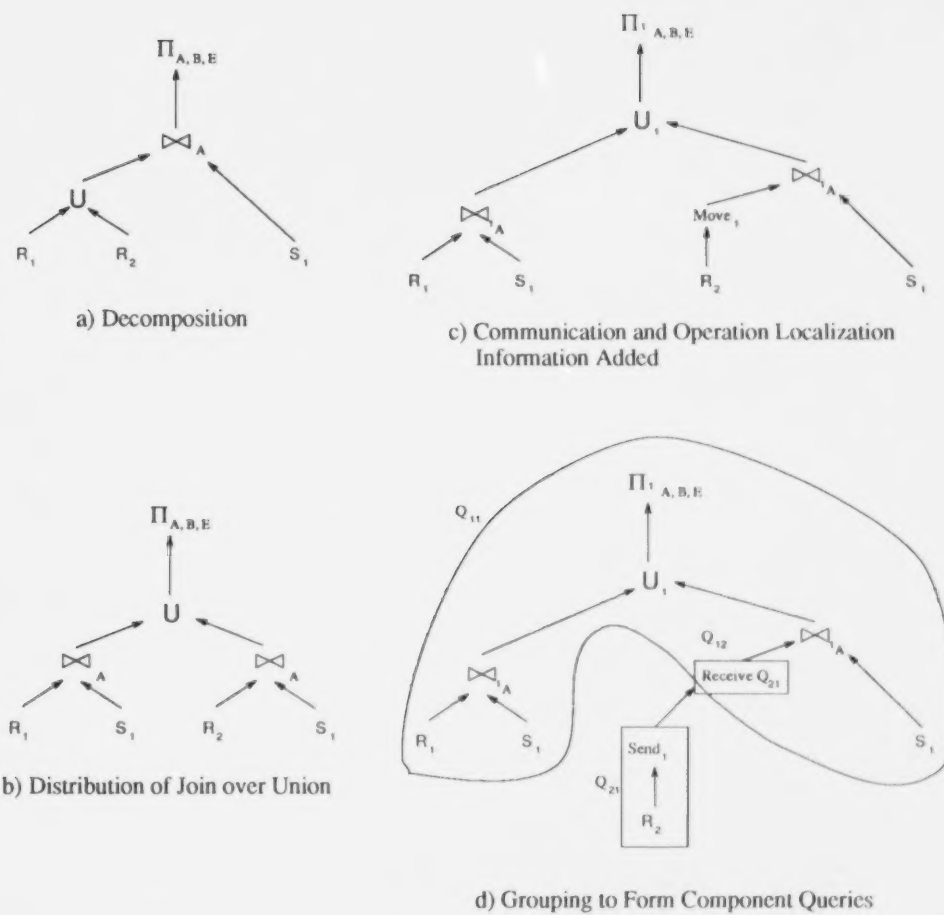


Figure 4.2: Algebraic Trees for Case 1

WHERE $T_{11}.A = S_1.A$

Code Fragment for Query Q_{12} This is executed at $CDBS_1$.

```
RECEIVE1 (Q21)
CREATE TABLE T11 (A, B, E)
INPUT (Q21) INTO T11
```

Code Fragment for Query Q_{21} This is executed at $CDBS_2$.

```
SEND1 (SELECT A, B, E
FROM R2)
```

In the code fragments that include the *receive* and *input* operations, there are different procedures to be executed depending on the kind of component database. The notation `CREATE TABLE Tij...` has been used although for secondary components this statement may only mean the creation of a file. That would be interpreted by the CIP depending on the capabilities of the CDB. In both cases 1 and 2, although processing of intermediate results is done at either component database, there is exploitation of parallelism. For example, in Figure 4.2.c, on the left bottom part of the tree, there is a join operation executed at CDB 1 ($\bowtie_{1c}$); at the same time, query Q_{21} can be executed at component DB_2 .

Case 2

In the second case, steps 1 and 2 are done in the same way as it was done in case 1. The difference comes at the third step when communication is involved at the optimization layer. In this case, it is preferable to process intermediate results at CDB_2 . The correspondent algebraic tree is shown in Figure 4.3.c. At this stage, as well as in case 1, the MDBQP generates the component queries that are the input to the CMDBSs, which are then translated into the component DB languages and then compiled at the CDBs.

The code fragments created to get the result query Q in this case are the following:

Code Fragment for Query Q_{11} . This is executed at CDBS 1.

```
SEND2 (SELECT A, B, E
FROM R1, S1
WHERE R1.A = S1.A)
```

Code Fragment for Query Q_{12} This is executed at C DBS 1.

```
SEND2 (SELECT A, B, E
FROM S1)
```

Code Fragment for Query Q_{21} This is executed at C DBS 2.

```
SELECT A, B, E
FROM T22, S2
WHERE T22.A = S2.A
UNION
T21
```

Code Fragment for Query Q_{22} This is executed at C DBS 2.

```
RECEIVE2 File (Q21)
CREATE TABLE T21 (A, B, E)
INPUT (Q11) INTO T21
```

Code Fragment for Query Q_{23} this is executed at C DBS 2.

```
RECEIVE (Q12)
CREATE TABLE T22 (A, B, E)
INPUT (Q12) INTO T22
```

4.4 Query Processing Phases

Query execution involves two phases: compilation which includes optimization and execution. It is, however, possible to perform optimization during both of these phases.

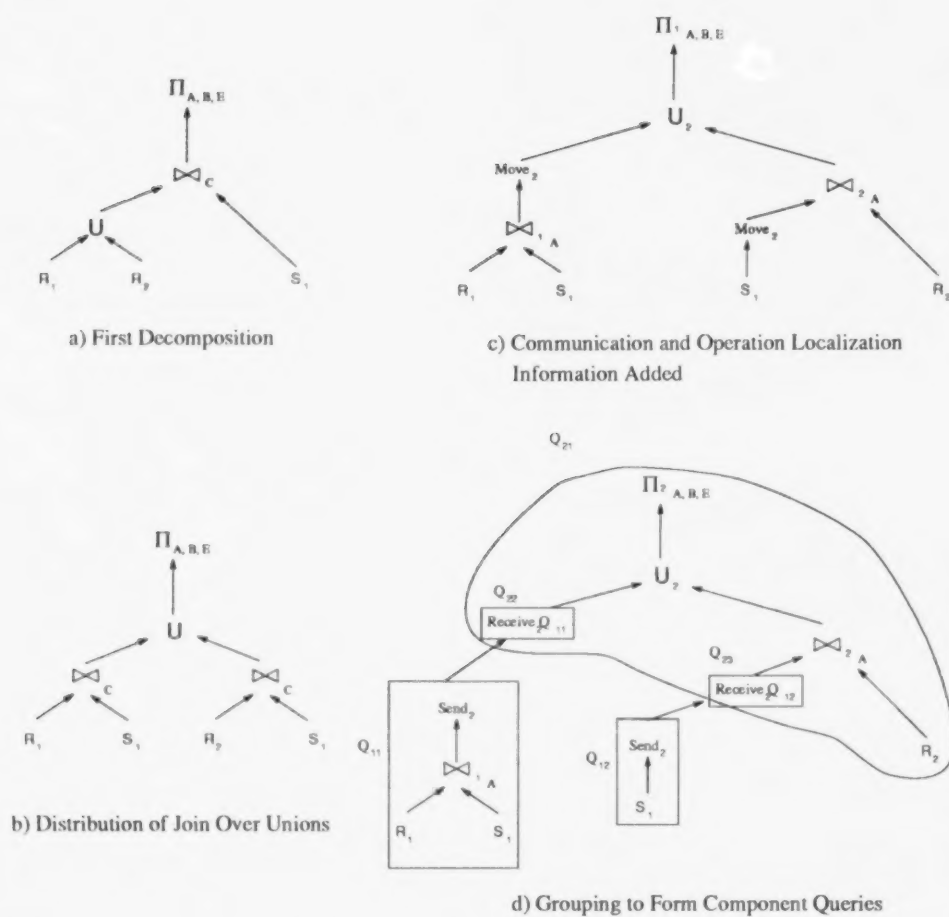


Figure 4.3: Algebraic Trees for Case 2

In [Zhu92], for example, an approach in which optimization is done at compilation as well as at execution time is presented. In our approach, we consider optimization only at compilation time since we are not treating the case in which queries make use of host variables that are known only at compilation time. In their approach, at compilation time, semantic query optimization and probing queries optimization are applied to the query whereas optimization at run time is done using parametric query optimization or adaptive query optimization. Semantic query optimization is done by transforming the original query into another semantically equivalent query. The transformations are done based on integrity constraints originally specified for global update operations. Probing queries make use of a separate statistic utility which collects information in the global catalogue. Such information is obtained by sending periodically probing queries to a set of multidatabase views. The use of probing queries is still under investigation, and it can be said that this procedure may become very expensive given that sending probing queries to the component databases involves not only processing time but also communication time. Parametric and adaptive query optimization are used at run time when host variables are known and the query may be further optimized.

4.4.1 Compilation Phase

The compilation phase tries to exploit compilation and optimization processes at components. As shown in Figure 4.4 there is no communication between component databases. This communication is not necessary since

the information about the components, relations and communications is kept in the MDB catalog which is accessible to the MDBQP but it is not accessible to the components. In the figure, the ovals indicate processes and the arrows indicate communication between processes.

During the compilation phase various pieces of code are stored in the MDB catalog, CMDBSs and the CDBs, to be invoked during the execution phase. The location of the code fragments is kept in the MDB Catalog and is accessible to the MDBQP.

In Figure 4.4, in the first process executed by the MDBQP, the query Q , i.e., participating relations, selection and projection attributes, etc., is entered into the process. Global query analysis, decomposition and optimization are performed here. The output is the different component queries that are the input to the CIP processes. Translations are done at this stage. The output is the component query, specified in the component query language. The CDBSs perform compilation and optimization of the correspondent queries.

4.4.2 Execution Phase

It is in the execution phase that intercomponent communication is needed, as indicated by the arrow between CIP processes in Figure 4.5.

The query Q and the respective bindings are entered to the first process, at the MDBQP. The MDBQP gets the location of the code fragments and fires them. The execution commands are issued and sent to the CDBSs. The CDBSs start executing the code obtained at compilation time. The CDBSs

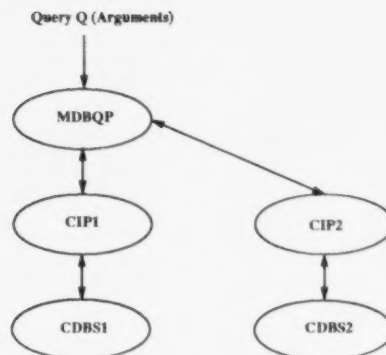


Figure 4.4: Compilation Phase

also keep control of the communication with other CDBSs as required, i.e. sending, receiving and storing temporary results. The component databases execute the component queries using the local cost functions, access query plan, etc. which are not under the MDB control.

There is no optimization at this phase since we are not considering binding of variables at execution time.

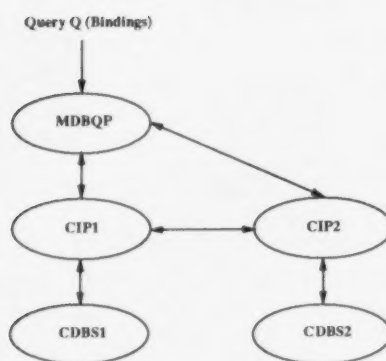


Figure 4.5: Execution Phase

Chapter 5

Optimization of Multidatabase Queries

To be able to do optimization, four different aspects of the problem have to be specified. They are:

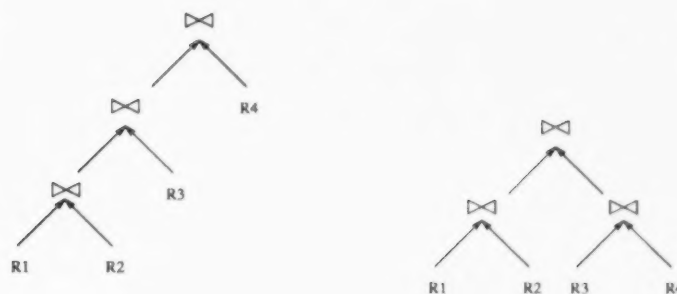
1. The search space or state space over which the search algorithm operates.
2. The transformation rules that determine the movement of the search algorithm from one state to another.
3. The specifics of the search strategy or algorithm.
4. The cost function that is applied to each state.

In this chapter we present the details of each issue mentioned above for the optimization problem in multidatabase systems.

5.1 Search Space

The search space consists of a family of equivalent (extended) relational algebra expressions denoted as processing trees. In these trees, the leaves are base relations, internal nodes are algebra operators, and the edges indicate the flow of data [Ull82]. Base relations are those at the CDBs. The trees that are going to be considered for optimization are those that contain *join* and *selection* operations. *Join* trees may be classified as *deep* or *bushy*. *Deep* or *linear* trees are those trees in which at each internal node, at least one of the children is a leaf (Figure 5.1.a) (or base relation), otherwise they are called *bushy* trees (Figure 5.1.b) [IK91]. The internal nodes are the join operation ($\bowtie$). Other algebraic operators may also be considered in future research. We have chosen to optimize join operations since it is the most expensive one. In this thesis, we are interested in optimizing *bushy* trees because, in a multidatabase environment, our objectives are two-fold: (1) do as much processing as possible at the components and (2) exploit as much parallel processing as possible. Bushy trees are the ones that express parallelism better. The decomposition process generates one of these trees which is the initial processing tree. The others are obtained by the application of transformation rules that will be discussed later and there is also a final processing tree which is the solution to the optimization of a query.

The initial translation gives an algebra expression on global relations only. Global relations are defined for the MDB and their fragmentation is transparent for the MDB user. Neither the fragmentation/localization information nor the transmission of intermediate results is incorporated in



a) Deep or linear query processing tree b) Bushy query processing tree
Figure 5.1: Types of Query Processing Trees

this tree. These are added in the decomposition and localization process, whose output is the initial tree or the first instance of an algebraic tree in the search space.

Localization and fragmentation information is then added to the tree. So the situation is the following:

For a multidatabase query Q , we have m component databases participating in query and n multidatabase relations like this:

$$\text{Relations} = \{R_1, R_2, \dots, R_n\}$$

Each $R_i \in \text{Relations}$ has the following fragmentation definition:

$$R_i = \cup_{j=1}^m R_{i,j}$$

where $1 \leq i \leq n$

For each relation R_i , there is at least one $R_{i,j}$ fragment. This ensures that there is at least one fragment for each relation, i.e., the relation is defined in the multidatabase.

Transformation rules can be applied to the algebraic trees in order to ob-

tain different solutions that yield, hopefully, a better cost. The tree obtained after applying transformation or reduction rules is a tree that is equivalent to the original tree. The transformation rules are the same as those in relational algebra. Also, reduction techniques are applied to the tree as specified in [ÖV91].

The *move* operator is introduced in between any branch of the tree, i.e., in between a relation (or fragment) $R_{i,j}$ or partial result $Q_{i,j}$ obtained using a relational algebra operator op (such as join, selection, projection, union, etc.).

The *move* operator does not necessarily have to be used in all instances. For example, if a relation or fragment is already residing at a primary component, there may be no need to move it to another component database. Even in the case that there is a relation or fragment at a secondary component, the move operator is not "needed", but the tree is a valid tree in the search space. The heuristics will determine which ones are going to be chosen as best alternatives. The move operator enables the transfer of data to the corresponding component databases so that the algebraic operations can be executed.

The state space is given by the transformed trees obtained when the transformation rules defined in the solution are applied. A valid extended relational algebra tree is the one that leads to a successful execution of the query. That means that the result of the query should be at the requesting component by the time the query is finished.

Finally, the goal state is that one in which the operations in the tree are in such distribution that they are grouped into as few component queries

as possible. This means that parallelism is exploited resulting in minimum response time and total cost. These can then be executed at the participating component databases.

5.2 Transformation Rules

In this section, we explain what the **transformation** functions in our optimization algorithm are. This function takes an algebraic processing tree (PT) as input, transforms it and generates an equivalent one. The transformation rules will be defined for the binary operator join ($\bowtie$). Other binary operations, such as union and cartesian product may also be considered in future research. A pre-optimization phase is assumed. In that phase, unary operators, such as selection (σ), are supposed to have been previously pushed down in the tree using other valid relational algebra rules [Ull82].

These rules are restricted by the *move* operator. As the move operator moves relations or temporary results only to primary components with temporary storage capabilities, then the transformations rules can be applied only when the localization and move operations on a relation allow us to do it.

Let us have the following component databases (CDBs):

- DB_i and DB_j are primary components with temporary storage capabilities.
- DB_k is a control database (primary component with temporary storage capabilities on top of which the MDBS resides).

- DB_i is a primary component without temporary storage capabilities or a secondary component.

Three relations are defined in the multidatabase, e.g., R , S and T . If the query $R \bowtie S$ is posed to the multidatabase, the alternatives to execute the query may be many. The valid transformations in the search space are now shown. The notation R_i indicates that relation R is located at CDB_i . To indicate that a join operation is executed at a particular CDB_i , the following is used, $\bowtie_i$.

It can be proven that the following general transformations (relational algebra rules) are applicable in the case of a multidatabase environment, in which the *move* operator is introduced for those relations that have to be transmitted to other databases, and the query is still a valid and executable one. The *move* operator must be attached to the relation and if the relation is affected by a transformation rule, the *move* operator remains attached to the relation. Following, the transformation rules (**TR**) are presented. The first four rules do not involve changes in the *move* operator. Rules 5 to 7 involve changes to the *move* operator.

TR1: Commutativity of binary operations. This rule applies to Cartesian product, union and join operations.

$$R \bowtie S \Leftrightarrow S \bowtie R$$

TR2: Associativity of binary operations. This rule applies to join and Cartesian product.

$$(R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)$$

It can be shown that for all cases of localization of R , S and T the rule is valid when introducing the *move* operator. For example:

$$(R_i \bowtie S_j) \bowtie T_i \Leftrightarrow (R_i \bowtie_i (Move_i S_j)) \bowtie_i T_i \Leftrightarrow R_i \bowtie_i ((Move_i S_j) \bowtie_i T_i)$$

TR3: Left join exchange. $(R \bowtie S) \bowtie T \Leftrightarrow (R \bowtie T) \bowtie S$

This can be deduced from the associativity rules.

TR4: Right join exchange.

$$R \bowtie (S \bowtie T) \Leftrightarrow S \bowtie (R \bowtie T)$$

The same as above.

There are a few rules concerned particularly with the move operator. In the following, we explain the cases that may appear when the operator is introduced. The transformation rules vary depending on the localization of the temporary relations.

Moving relations or fragments to non-relational databases may incur higher processing and translation costs. The cases treated here are the most general ones, i.e., those in which there are primary and secondary components present in the multidatabase. There may be also primary components at which it is not possible to save the partial results obtained as a result of a query that needs further processing. Also there is a control database at which queries may be processed. It is assumed that this database has temporary processing and storage capabilities so that if it is not possible to execute a query at any of the components, the control database may be used.

TR5: $R_i \bowtie S_j \Leftrightarrow R_i \bowtie_i (Move_i S_j)$

$$R_i \bowtie S_j \Leftrightarrow (Move_j R_i) \bowtie_j S_j$$

$$R_i \bowtie S_j \Leftrightarrow (Move_k R_i) \bowtie_k (Move_k S_j)$$

This rule shows that if the relations are located at two different primary components, i.e., CDB_i and CDB_j , with temporary storage and processing capabilities, the valid transformations are to execute the query at either of them or at the control database, CDB_k .

TR6: $R_i \bowtie S_l \Leftrightarrow R_i \bowtie_i (Move_i S_l)$

$$R_i \bowtie S_l \Leftrightarrow (Move_k R_i) \bowtie_k (Move_k S_l)$$

If one of the participating relations is located at a secondary component CDB_l (or primary without temporary storage capabilities) and the other one at a primary component with temporary storage capabilities CDB_i , then the processing may be done at the primary component CDB_i or at the control database CDB_k .

TR7: $R_l \bowtie S_l \Leftrightarrow (Move_i R_l) \bowtie_i (Move_i S_m)$

$$R_l \bowtie S_l \Leftrightarrow (Move_k R_l) \bowtie_k (Move_k S_m)$$

R and S may be located at different secondary databases or primary databases without temporary storage capabilities CDB_l and CDB_m and the same rules apply. In this case, the choice is to execute the query at the control database CDB_k . The other alternative is to move both relations to a primary component with temporary storage capabilities CDB_i which is not involved in this join, but that may accept to store both relations and execute the query there.

5.3 Search Strategy

The search strategy may vary depending on the number of join operations that are in the query. The search strategy may be *enumerative* or *randomized* [LV91]. Enumerative search strategies provide the same answer whenever the search algorithm is applied. Besides, most enumerative search strategies search almost through the whole search space, pruning only some of the possible solutions. They are, therefore, very likely to find the optimal solution. Randomized strategies search for the solution around some particular point. The answer will depend on the initial state chosen and may vary for different searches through the same search space. Randomized techniques may not find the optimal solution but they reduce the cost of optimization. It should be mentioned that the search space of a query optimization is a combinatorial problem depending on the number of joins in the query [Swa89]. It has been shown, for a centralized environment, that *randomized* algorithms may give the optimal answer in shorter time than exhaustive search in most of the cases for queries which contain a large number of join operations (> 10) [IW87], [IC90], [SG88], and [Swa89]. The *enumerative* strategies may be further classified as *exhaustive search* and *branch and bound* or *use of heuristics*. We will focus on the use of heuristics and will leave the use of randomized techniques as a further research topic. Since the number of joins in a query determines the size of the search space and the queries that were tested had at the most 8 joins, it was decided that the use of heuristics would suffice. Besides, the heuristics used emphasize on the search through bushy trees only, which greatly reduces the search space.

Algorithm 1:**Input:** An initial processing tree (PT)**Output:** An optimal execution plan for query Q

```
begin
  PT := setInitState();
  while not stopCond()
  begin
    chooseRule (PT)
    PT1 := transform (PT);
    if acceptTransform (PT,PT1) then
      begin
        PT := PT1;
      end;
    end;
  end;
  return (PT)
end
```

Figure 5.2: Enumerative Search Algorithm-Using Heuristics

The search strategy is very much related to the transformation rules, since the former makes use of the latter. It is assumed that there will be an initial state which will be the input to the query optimization phase. Then, as shown in the algorithm of Figure 5.2, the *choose rule* function is used, e.g., the rules are applied in order to the same node, the order is first the commutativity rule and then the rules related to the *move* operator. A rule is applied and the cost of the new PT (PT1) is computed and compared to the previous lowest cost (of PT). Otherwise, the PT with lowest cost remains in PT. Those are the steps of *acceptTransform*. This process is repeated until the rules have been applied to every node of the tree.

The heuristics applied in our methodology are the following:

1. Only *bushy* trees are considered. By eliminating linear trees from the search space, we are also *pruning* it, since we are discarding those possibilities. They are discarded mainly because, as stated before, *bushy trees* represent more parallelism than linear trees because operations are spreaded all over the branches. That is why the rules for associativity and left and right exchange are not applied. They might generate linear trees. These rules may be implemented later and added to the current implementation to experiment and observe the results and compare them to the bushy trees.
2. For a particular node, a rule may be applicable and more than one option is possible. Only one of them is going to be taken into account. For example, if the component databases involved in a query are primary components with temporary storage capabilities, the data is going to

be moved to either of them. The option of sending them to the control component database will be discarded. This option will be taken only when strictly necessary, i.e., both participating components are secondary or primary without temporary storage capabilities.

5.4 Cost Functions

The cost functions are used to calculate the cost of executing a particular execution plan represented as an algebraic processing tree. These functions (both for response time and for total time) are defined over the nodes of the tree and are recursive in nature. They are calculated while the tree is traversed and there is a cost weight associated with moving from a node to its children, i.e., the cost of that arc.

Let us assume that the multidatabase MDB has CD number of component databases and TR total number of relations.

5.4.1 Response Time

Any query tree in the search space of a multidatabase query has the following components:

N participating relations¹. These are the leaves of the tree. Also,

$1 < N < TR$.

J join operations in the tree

¹The participating relations at the global level may become fragments when the fragmentation schema is used to add the localization information to the tree. Therefore when we say relations, we mean either base relations or fragments.

M participating databases, and $1 < M < CD$.

The computation of the response time cost of a particular processing tree, PT, is recursive in nature. This is because the cost of a node has to be determined in terms of the cost of its successors (or sons), whose cost also depends on their successors and so on up to the leaves of the tree. Since the purpose of calculating response time cost is to find the longest sequential time to retrieve the answer to the query, the computation of the cost is the most costly path in the tree from root to leaves. Therefore, the tree has to be traversed recursively, calculating the cost at each node in terms of the cost of its sons.

Such a function is the following:

$$resp\_time(n) = \begin{cases} 0 & \text{if } n = \text{leaf} \\ cp(n) + \max(resp\_time(n.sons)) & \text{if } n = \text{inner} \end{cases}$$

n is any node of the PT.

$n.sons$ are the nodes which are sons of node n .

$cp(n)$ is the processing cost² of node n .

For terminal nodes (leaves) the processing cost is zero because they are the base relations or fragments.

The inner nodes are algebraic operations. These operators may execute on one or more relations. In the case of unary operators, the list of sons or $n.sons$ is composed of only one node. Therefore the maximum cost is the cost of the single son. These operators may be projection or selection. When the operator is binary, the sons may be two or more. In this case, the maximum cost is added to the cost of processing at node n . This is done

²In section 5.4.3 we show how the processing cost of node n is obtained.

recursively until the leaves of the tree are reached.

If node n in the function is the root of the tree, the result of evaluating the function gives the response time cost for that PT.

5.4.2 Total Time

For the total time, the cost function can be expressed also in a recursive manner. The total time cost of a tree is the sum of the cost of all nodes of the tree. Therefore, we get the following function:

$$tot\_time(n) = \begin{cases} 0 & \text{if } n = \text{leaf} \\ cp(n) + \text{sum}(tot\_time(n.sons)) & \text{if } n = \text{inner} \end{cases}$$

The processing cost of the leaves is zero since the leaves of the tree are base relations or fragments.

For the inner nodes, i.e., those nodes that represent the join operations, the cost is given by the sum of the cost of the sons. The calculation of the cost of each node is done by adding the cost of processing the operation at the specified component database, the cost of communication between the node and its sons and the cost of storing the intermediate results.

The way in which the cost functions have been specified affects the transformation rules that are going to be used to get new equivalent trees. This is because the values of the cost functions depend on where the relations are going to be processed and the transformations should get different configurations of localization of relations and intermediate results. These may also be accompanied by join transformations rules. The rules are fully specified in the section corresponding to the transformation rules.

5.4.3 Component Cost Determination

The cost of executing the node operation at a component database is represented by $cp(n)$ in the cost functions. If the node is an inner node to which data has been transferred from another database, $cp(n)$ is composed of the cost of processing the subquery, the cost of transmitting the data from the other databases and the cost of storing the data.

Therefore, we have that $cp(n) = cp\_c(n) + c\_c(n) + ts\_c(n)$

where $cp\_c(n)$ is the component processing cost of node n ,

$c\_c(n)$ is the communication cost of node n and

$ts\_c(n)$ is the temporary storage cost of node n .

The approximation of the communication or transmission cost is done by using a linear cost function that depends on the amount of data transmitted between a child node and its parent node. Hence, we have the following linear communication cost function: $lcc_{ij}(x)$ where x is the number of bytes transmitted from CDB_i to CDB_j and it is defined as follows:

$$lcc_{ij}(x) = C_{msg} + C_{tr} * x$$

where C_{msg} is the fixed cost of initiating and receiving a message and

C_{tr} is the cost of transmitting a data unit from one component to another.

When considering $c\_c(n)$, the cost is calculated by adding the cost of transmitting the data from the sons of node n for processing at that component. The linear function $lcc_{ij}(x)$ where x is the number of bytes transmitted is defined for each pair ij of component databases present in the multidatabase. That information is kept in the multidatabase dictionary.

Therefore if the sub-query at node n is executed at component k and data

is transmitted to k from the two children of node n which are at components l and m , the amount of data transmitted from l to k is x , the amount of data transmitted from m to k is y , the communication cost between k and l is $lcc_{lk}(x)$ and the communication cost between k and m is $lcc_{mk}(y)$, then $c.c(n)$ is given by $lcc_{lk}(x) + lcc_{mk}(y)$.

$ts.c(n)$ can be determined using information from the component databases on the creation of temporary tables. It is the sum of the creation time and the time to insert the rows of that table. This cost depends also on the number of tuples of the tables being stored temporarily and the tuple size. It may be computed using *calibration* as it is done for the cost of processing. *Calibration* is a methodology used to predict the cost of processing at CDBs. Therefore, $ts.c(n)$ is a function $tsc(x, y)$ where x is number of tuples to be stored and y is the size of the tuples.

The functions for the cost of processing the subquery at a component are shown below. Values for a particular calibrated database are also given.

In this subsection, we explain how the value of processing the query at a component database is calculated for *conforming DBMSs*.

The methodology used for calculating the processing cost at the CDBs is that one defined in [DKS92]. This methodology deduces necessary information for the cost functions by calibrating a given DBMS.

Conforming DBMSs, as shown in [DKS92], are those CDBMSs which are from different vendors and do not provide the cost functions used but give some database statistics. These statistics are accessible to the multidatabase DBMS and therefore may be used to predict the value of the cost functions.

A new methodology for predicting the cost of processing at CDBs has

been proposed in [DKS92]. This methodology makes use of an artificially generated database (the *calibrating* database) to predict the cost of other queries posed to the same type of database. The database calibration shown in [DKS92] is in terms of response time. This time can be estimated in terms of the initialization cost, the cost to find the qualifying tuples and the cost to process the selected tuples. These cost functions depend on the type of query (selection, projection or join) and on the method used to perform the query. The methods considered for selection are sequential scan, index-only scan, clustered index scan and unclustered index scan. For join, the methods are nested loop, and ordered merge.

The calculation of the cost function coefficients is done using a calibrating database and its properties. Such database is built by generating values for each attribute. Each column is assumed to use one of the access methods mentioned above, e.g., indexed clustered, unclustered, etc. The calibrating procedure consists of posing queries to the calibrating database and deducing the coefficients of the cost functions.

The coefficients for three different DBMSs are given in [DKS92]. The DBMSs that were tuned are Allbase, DB2 and Informix. The results for Oracle are not in the paper, but according to the authors, the results for Oracle are very close to those of Informix [Du92]. Version 6.0 of Oracle was used to do the calibration. After the determination of the coefficients, one of them is still variable. This variable one is different from the rest, which are constants. The variable coefficient is dependent on the tuple size of the relation for the selection case. All the information needed for doing such calculations is kept in the data dictionary or can be obtained from the query

specifications when being optimized. The functions and the calibrated values for an Oracle instance are shown in the next subsection.

The value of the coefficients should be found for every different type of attribute. The values obtained for the case of integer attributes is shown in [DKS92].

The functions for estimating the cost are categorized for selection and for joins. As mentioned already, the methods considered for selection are sequential scan, index-only scan, clustered index scan and unclustered index scan. For join operations, they are nested loop (if there is sequential scan on the second relation in the join) and ordered merge.

For determining the value of the coefficients for the formulae for selection, the access methods used on each column of the relations used in the calibrating database are found based on the particular properties of this database. Then, a group of queries is issued and the values are calculated. Therefore, the relations used for calibration are not necessary after finding the coefficients [DKS92]. The coefficients for joins may then be calculated since the join formulae are defined in terms of selection operations.

Calibrated Values

Let us now have a look at an example in which the tuned values are used and the functions are evaluated. As it was stated previously, only the values for integer attributes in the query selection are presented in the paper. Therefore, in the example, the attributes of the relations are assumed to be all integer type.

Let us assume we have two relations R_1 and R_2 with sizes N_1 and N_2 respectively. The cost is estimated for two operations: a select operation on R_1 with selectivity S_1 and a join operation on R_1 and R_2 with selectivity J_{12} . This data is part of the statistic information kept in the data dictionary of the MDB.

After doing the tuning, the functions obtained for an Oracle database are the following:

- Cost of Selection using Sequential Scan CS_{ss} :

$$CS_{ss} = CS0_{ss} + ((CS1_{ss}^{io} + CS1_{ss}^{cpu}) * N_1) + (CS2_{ss} * N_1 * S_1)$$

$$CS_{ss} = 0.06 + (((168 + ts)/7 * 10^5) * N_1) + (0.00045 * N_1 * S_1)$$

- Cost of Selection using Index-Only Scan CS_{is} :

$$CS_{is} = CS0_{is} + CS1_{is} + (CS2_{is} * N_1 * S_1)$$

$$CS_{is} = 0.006 + 0.001 + (0.001 * N_1 * S_1)$$

- Cost of Selection using Clustered Index Scan CS_{ci} :

$$CS_{ci} = CS0_{ci} + CS1_{ci} + (CS2_{ci} * N_1 * S_1)$$

$$CS_{ci} = 0.006 + 0.001 + (0.001 * N_1 * S_1)$$

- Cost of Selection using Unclustered Index Scan³ CS_{ui} :

$$CS_{ui} = CS0_{ui} + CS1_{ui} + (CS2_{ui} * N_1 * S_1)$$

$$CS_{ui} = 0.006 + 0.001 + ([0.001, 0.002] * N_1 * S_1)$$

³The value of $CS2_{ui}$ varies slightly on table size.

When the previous values are obtained for two particular relations R_1 and R_2 , the join functions can be evaluated as follows ⁴:

- Cost of Join using Nested Loop CJ_{nl} : (if sequential scan on R_2)

$$CJ_{nl} = CS_{xx}(R_1) + CS_{0ss}(R_2) + CS_{1ss}^{io}(R_2) + (N_1 * S_1 * (CS_{ss}(R_2) - CS_{1ss}^{cpu}(R_2)))$$

- Cost of Join using Nested Loop CJ_{nl} : (if index scan on R_2)

$$CJ_{nl} = CS_{xx}(R_1) + CS_{0ss}(R_2) + (N_1 * S_1 * (CS_{ss}(R_2) - CS_{0ss}(R_2)))$$

- Cost of Join using Ordered Merge CJ_{om} :

$$CJ_{om} = CJ_{1or}(R_1) + CJ_{1ss}(R_2) + CS_{ss}(R_1) + CS_{ss}(R_2) + CJ_{2mg} * N_1 * N_2 * J_{12}$$

The selectivity factors are kept in the multidatabase dictionary and they can be accessed to calculate the cost of the join and selection operations.

The proprietary DBMS case is still under research but may be incorporated later into the model.

⁴The subindex xx is to show that any selection method can be used.

Chapter 6

Experimentation and Results

To validate our approach, the query optimizer was implemented and experiments were run to observe the behaviour of the optimizer and its effects. In this chapter, we present the experimental setup in which the query optimizer is tested, the instrumentation used, the experiments that were run and the results that were obtained.

6.1 An Actual Multidatabase System

In this study we use a real-life system from which we draw our data and queries. The Land-Related Information Systems (LRIS) Network is a multidatabase system which gives access to three component databases managed by the Land Information Services Division of Alberta Forestry, Lands and Wildlife [Alb92]. The component databases are the following:

1. *Alberta Land Titles Automation (ALTA)*. This database maintains information about the land titles in the province of Alberta, Canada. This is a primary database, i.e., a relational one and it has temporary storage capabilities. Information about the title references is kept as well as information about the title holders.
2. *Land Status Automated System (LSAS)*. This database contains information about the administrative status of the land and the clients that are performing activities on that land. For our purposes, this is a relational database system (primary database) and it does not have secondary storage capabilities.
3. *Land Survey Document System (LSDS)*. This database contains information about the plans that are taking place at a particular piece of land and also information about the municipality to which it belongs. This is a relational database in our multidatabase. It has secondary storage capabilities.

The LRIS system allows users to access data from the three component databases by means of a global schema which makes the components invisible to the user. The global schema is presented in Figure 6.1. Dotted lines mark the boundaries of each component database. The schema diagram which shows the relations and their relationships is presented¹. In the diagram, boxes represent relations and lines show relationships between them.

¹The diagram has been simplified to show only the necessary relations for the examples in this work. This applies to the components and the multidatabase system.

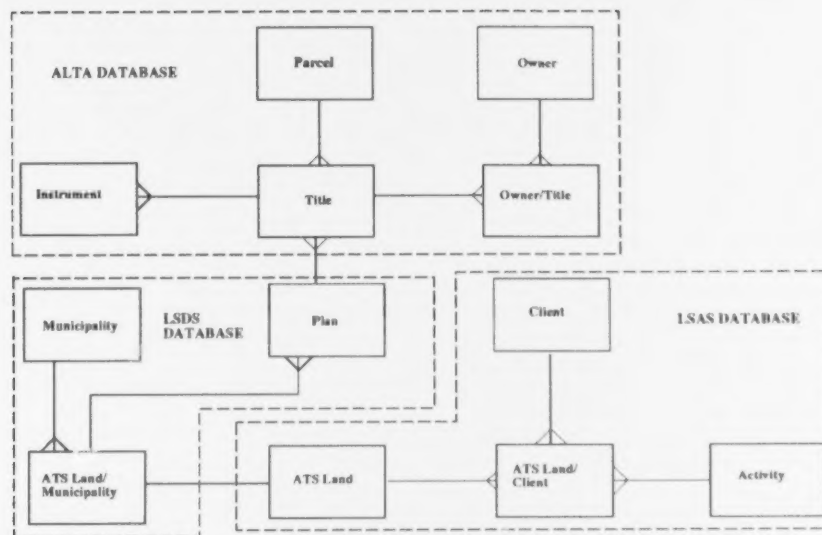


Figure 6.1: Multidatabase

The multidatabase is implemented as a relational database composed of twelve relations. Thus, this is an example of a homogeneous multidatabase system. The description and attributes of these relations are given in Appendix A.

6.2 Implementation

The implementation of the core part of the optimizer has been written in two languages. The optimization algorithm is written in Quintus Prolog Release 3.1.1 and the ORACLE C Precompiler Version 1.4.8.2.2. The Prolog program makes calls to the Pro*C subroutines to access the global data dictionary that is kept in an Oracle database.

The algorithm is coded in Quintus Prolog since Prolog provides means for easy rule application and tree manipulation. The representation of the tree is done by having each node of the tree represented as a *fact* which has the information about its sons, whether it is an operator or a base relation or fragment, the component database associated with the operation (where it is going to be executed), the localization of the relation (fragment), and an indicator of the database to which this result is moved after processing.

For calculating total time cost, the tree is traversed in pre-order. A call to a Pro*C function is issued for every node of the tree. Response time cost is calculated by traversing the tree depending on which son has the highest cost of all sons of a node. Again, a call to a ProC function is issued for calculating the cost.

The routines in ProC calculate the values needed for the cost functions, i.e., response time and total time cost. These are based on the statistics contained in the data dictionary as shown in the corresponding section of this thesis (Section 6.3). Besides the processing cost, the communication cost and temporary storage cost is calculated. For these experiments, some data, such as initial transmission cost, needed for calculating the cost, are estimated values. Such data was not available from the source.

The rules are implemented by matching the head of every rule with the node being examined. If the head matches, then the conditions for the rule are examined. If the conditions are satisfied, the new node(s) is asserted and the old ones are retracted. This is done for all the rules.

6.3 Experimental Setup

6.3.1 Metrics

There are two measurements that are of interest: measuring the effectiveness of the optimization methodologies and measuring the effects of the multi-database environment. Our optimization methodology is heuristics-based. We, therefore, test its effectiveness in finding the optimal solution. We claim that the environment, particularly the autonomy of the component databases, determines what the optimal solution is. We have experiments to show how this issue influences the response time and total time cost.

For the methodology, we measure the variation in total and response time cost depending on the rules used. The heuristics that determine the application of rules are varied and their effects on estimated cost is measured. This measures the effectiveness of the rules independently from each other. One rule is applied at the time to every node of the tree being optimized.

To measure environmental effects, three experiments are run. These are:

1. Measure the effects of the autonomy and capabilities of the component databases. This is also a measure of the effect of the *move* operator.
2. Measure the effect of the localization of relations. The relations may be located at different component databases.
3. Measure the effect of the size of the participating relations, and see how the cost varies for response time cost as well as for total time cost.

6.3.2 Instrumentation

The main objective of the study is to optimize operations that execute over multiple CDBMSs. The most important of these operations is join (denoted $\bowtie$), so the test queries involve joins over multiple CDBMSs.

Experiments involving other algebraic operations may be tested later and compared to these results to see how much effect they have on performance.

The queries posed to the multidatabase system access data from the three CDBs. Mostly, there are join operations that link data from one database to another. We use the following queries posed over the LRIS multidatabase described earlier:

1. Find the titles that have plans registered for them. This query accesses two relations on different CDBs and involves one join.

```
SELECT PLAN.Plan_Reg_Num, TITLE.Tit_Ref_Num
FROM PLAN, TITLE
WHERE PLAN.Plan_Reg_Num=TITLE.Plan_Reg_Num
```

2. Find the titles that have plans registered for them and the instruments associated to every title. This query accesses three relations on two CDBs and involves two joins.

```
SELECT PLAN.Plan_Reg_Num, TITLE.Tit_Ref_Num, INS.Id
FROM PLAN, TITLE, INS
WHERE PLAN.Plan_Reg_Num=TITLE.Plan_Reg_Num and
      TITLE.Tit_Ref_Num=INS.Tit_Ref_Num
```

3. For a particular client, find the pieces of land on which the client is doing some activity. Include the title information for the pieces of

land. This query accesses four relations on three component databases and involves three joins.

```
SELECT CLI.Cli, ATS.ATS_Id, PLAN.Plan_Reg_Num
FROM ATS, ATS_MUNIC, PLAN, TITLE
WHERE ATS_MUNIC.ATS_Id=ATS.ATS_Id and
      ATS.ATS_Id=PLAN.ATS_Id and
      PLAN.Plan_Reg_Num=TITLE.Plan_Reg_Num
```

4. For a particular title, find the activities that are taking place in that piece of land. This query accesses five relations on three component databases and it involves four joins.

```
SELECT TITLE.Tit_Ref_Num, ATS_CLI.Cli_Id,
      ATS.ATS_Id, PLAN.Plan_Reg_Num, INS.Id
FROM TITLE, PLAN, ATS, ATS_CLI, INS
WHERE TITLE.Plan_Reg_Num=PLAN.Plan_Reg_Num and
      PLAN.ATS_Id=ATS.ATS_Id and
      ATS_CLI.ATS_Id=ATS.ATS_Id and
      TITLE.Tit_Ref_Num=INS.Tit_Ref_Num
```

5. Find all the plan registration numbers for land, their title holders and find which type of ownership they have. This query accesses six relations on three component databases and it involves five joins.

```
SELECT ATS.ATS_Id, TITLE.Tit_Ref_Num,
      CLI.Cli_Id, CLI.Name,
FROM TITLE, PLAN, ATS, ATS_MUNIC, ATS_CLI, CLI
WHERE TITLE.Plan_Reg_Num=PLAN.Plan_Reg_Num and
      PLAN.ATS_Id=ATS.ATS_Id and
      ATS_MUNIC.ATS_Id=ATS.ATS_Id and
```

```

ATS.MUNIC.Munic=MUNIC.Munic and
ATS.CLI.Cli_Id=CLI.Cli_Id

```

6. Find all the information as in query 3 including also the description of the instruments. This query accesses eight relations on three component databases and it involves six joins.

```

SELECT ATS.ATS_Id, TITLE.Tit_Ref_Num,
       CLI.Cli_Id, CLI.Name,
       MUNIC.Munic, MUNIC.Descrip,
       INS.Ins_Id, INS.Ins_Type, INS.Ins_Text
FROM TITLE, PLAN, ATS, ATS_MUNIC,
     MUNIC, ATS_CLI, CLI, INS
WHERE TITLE.Plan_Reg_Num=PLAN.Plan_Reg_Num and
      PLAN.ATS_Id=ATS.ATS_Id and
      ATS_MUNIC.ATS_Id=ATS.ATS_Id and
      ATS_CLI.Munic=MUNIC.Munic and
      ATS_CLI.Cli_Id=CLI.Cli_Id and
      TITLE.Ref_Num=INS.Ref_Num

```

7. Find all the information as in query 3 including also instrument information but excluding owner information. This query accesses eight relations on three component databases and it involves seven joins.

```

SELECT ATS.ATS_Id, TITLE.Tit_Ref_Num,
       CLI.Cli_Id, CLI.Name,
       MUNIC.Munic, MUNIC.Descrip,
       INS.Ins_Id, INS.Ins_Type, INS.Ins_Text,
FROM TITLE, INS, PLAN, ATS, ATS_MUNIC,
     MUNIC, ATS_CLI, CLI

```

```

WHERE TITLE.Ref_Num = OWN_TITLE.Ref_Num and
      TITLE.Plan_Reg_Num=PLAN.Plan_Reg_Num and
      PLAN.ATS_Id=ATS.ATS_Id and
      ATS.MUNIC.ATS_Id=ATS.ATS_Id and
      ATS.CLI.Munic=MUNIC.Munic and
      ATS.CLI.Cli_Id=CLI.Cli_Id and
      TITLE.Ref_Num=INS.Ref_Num

```

8. Find all the information as in query 3 including also parcel information. This query accesses ten relations on three component databases and it involves eight joins.

```

SELECT ATS.ATS_Id, TITLE.Tit_Ref_Num,
      OWN.Own_Id, OWN.Name, OWN.Occupat,
      CLI.Cli_Id, CLI.Name,
      MUNIC.Munic, MUNIC.Descrip,
      PAR.LINC_Num, PAR.Short_Leg,
FROM TITLE, OWN_TITLE, OWN, PLAN, ATS,
      ATS_MUNIC, MUNIC, ATS_CLI, CLI, PAR
WHERE TITLE.Ref_Num = OWN_TITLE.Ref_Num and
      OWN_TITLE.Own_Id = OWN.Own_Id and
      TITLE.Plan_Reg_Num=PLAN.Plan_Reg_Num and
      PLAN.ATS_Id=ATS.ATS_Id and
      ATS_MUNIC.ATS_Id=ATS.ATS_Id and
      ATS_CLI.Munic=MUNIC.Munic and
      ATS_CLI.Cli_Id=CLI.Cli_Id and
      TITLE.LINC_Num=PAR.LINC_Num

```

Spooled Output

For: Ana Dominguez

UID: ana@sunwapta.cs.ualb

Pages: 76

JobName: (stdin

Date: Thu Oct 7 23:59:45 1993

Printer: LW776 :LaserWriter@*

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Ana M. Domínguez

TITLE OF THESIS: Query Optimization in Multidatabase Systems

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1993

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed)

Ana M. Domínguez
102, 12215 Lansdowne Dr.
Edmonton, Alberta
T6H 4L4

Date:

Information about the LRIS database is used by the MDBS query optimizer. The relevant information is kept in the MDB Catalog as relations. These include the following:

Relation Name: COMPONENTS

This table contains information about the component databases and their capabilities. The attributes of this table, their types, and sizes are shown in Figure 6.2. The DB_TYPE shows whether a component is a primary (1) or secondary (2) component database. Zero (0) is used for the control database which is always a primary database.

Relation Name: MDB_TABLES

This table contains information about the relations (tables) that are defined in the multidatabase. This information includes the localization of each relation in the underlying component databases. The relations are those described in Section 6.1 and Appendix A and are not repeated here. Relation sizes or cardinalities and selectivity factors are given in Figure 6.3. The selectivity factors are for those attributes of the relations that are used in the joins. The attributes of this table are shown in Figure 6.2.

Relation Name: COLUMNS

This table contains information about the attributes of the relations defined at the multidatabase level. This information is also given in Section 6.1 and Appendix A. The attributes of this table are shown in Figure 6.2.

Relation Name: COST_INFO

This table contains information that are obtained using the methodology developed by the Pegasus project [DKS92] about the tuning of the databases. The information includes the value of the coefficients for the cost functions,

COMPONENTS					
COMPONENT_DB	Number		1	2	3
COMPONENT_NAME	Char	20	ALTA	LSAS	LSDS
DB_TYPE	Number		Cont (0)	Prim (1)	Prim (1)
TEMP_ST_CAP	Char	1	Yes (1)	No (0)	Yes (1)

MDB TABLES		
TABLE_NAME	Char	20
COMPONENT_DB	Number	
CREATOR	Char	20
TABLE_TYPE	Char	1
TABLE_ID	Char	10
CARDINALITY	Number	
TUPLE_SIZE	Number	

COLUMNS		
TABLE_NAME	Char	20
COLUMN_NAME	Char	40
DATA_TYPE	Char	20
DATA_LENGTH	Number	

COST_INFO	
COMPONENT_DB	Number
CS0SS	Number
⋮	
CS2UI	Number

Figure 6.2: Attributes of the Multidatabase Catalog Tables

Table Name	Cardinality	Selectivity
Client	41,750	0.00002395
ATS Land	1,635,000	0.00000061
ATS Land/Client	1,500,000	0.00000061
Activity	178,000	0.00000056
Title	1,000,000	0.00001000
Parcel	1,000,000	0.00001000
Owner	800,000	0.00125000
Instrument	100	0.01000000
Owner/Title	1,500,000	0.00000066
ATS Land/Municipality	6,531	0.00001395
Municipality	74	0.01351350
Plan	215,000	0.00004650

Figure 6.3: Table Cardinalities

so that the global cost functions can be evaluated. Since it is supposed that the components are homogeneous and relational, the calibrated values of the coefficients are taken as those presented in [DKS92] which are presented in this thesis in Section 5.4.3. The attributes of this table are shown in Figure 6.2 and their cardinalities are given in Figure 6.3².

The underlying network is assumed to be a very high speed network known as Broadband ISDN and ATM network³. The values used are 10^{-6} secs for the initial transmission cost (C_{msg}) and a transmission rate of 10^{12} bits/sec (C_{tr} is 10^{-12}) [BG92].

²The value shown for the table ATS Land/Client is an estimated value and it might not be the most accurate one.

³ISDN stands for Integrated Services Digital Network and ATM stands for Asynchronous Transfer Mode.

6.4 Experiments and Results

In this section, the dependent and independent variables for each experiment are described. Then, the results obtained from each experiment are presented and analyzed.

In the remainder, whenever we mention *cost*, we are referring to the cost of executing a query in terms of time. The time is always measured in seconds. The two measures used are *response time cost* which is the time that the query takes to be executed from the time it is issued to the time the answer is provided and *total time* which is the total time spend executing the query, including the processes that are done in parallel at different component databases. When we refer to the *number of joins*, we mean the number of joins of base relations and not the joins of intermediate results.

The values of response time and total time cost shown are estimated. It would be useful to implement the complete query processor in order to get the real values of performing the queries. Such experimentation is left for further research.

6.4.1 Effect of Rules

The same set of queries are optimized, changing the rules used.

The independent variable is *number of joins in each query* for optimization. The dependent variable is the *cost* of the optimized queries. The rules are applied one at a time to the whole tree. The results are shown in Figures 6.4 for response time cost and in 6.5 for total time cost.

When applying rules separately, it is important to notice that the differ-

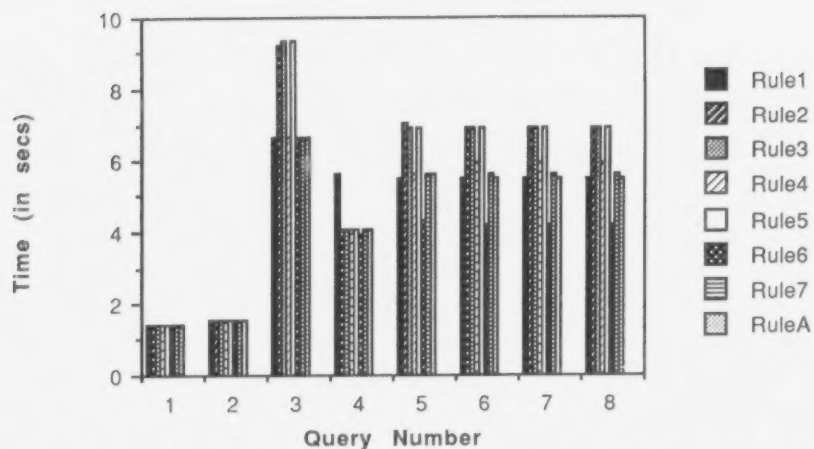


Figure 6.4: Different Sets of Rules by Priority (Response Time Cost)

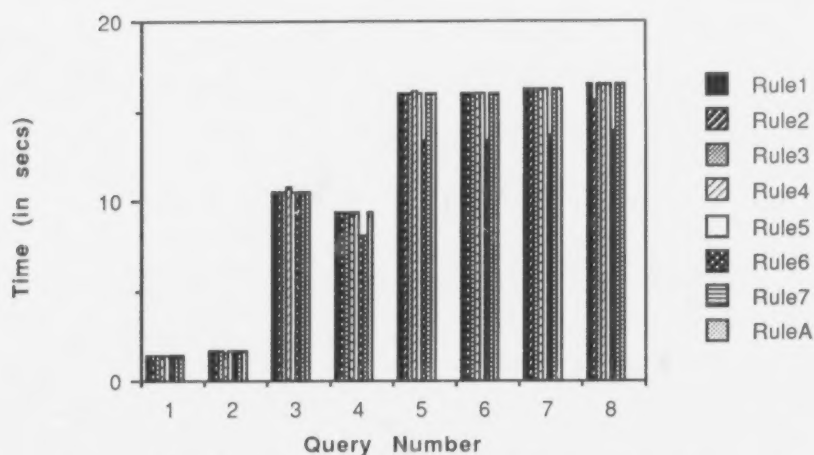


Figure 6.5: Data for Different Sets of Rules (Total Cost)

ence in cost between the curves increases as the number of joins increases. This shows that although the rules related to the move operator did not get a better result than the commutativity rule for small queries, it gets better results for large queries. This may be due to the greater exploitation of parallelism in large queries, particularly in communication cost. Although the difference is not very visible between rules 1, 5 and 6, it is obvious between rules 1 and 7, and this result is the same for both, response time as well as total time.

In Figure 6.4 for response time, we can see that there is a peak value when the number of joins is three. This is due to the fact that the tree is not completely bushy because of the number of joins in the query. We can also see a peak in the total time cost curves, Figure 6.5, when the number of joins is three. This may be due to the fact that the relations considered in the query are quite big, having the highest number of tuples.

6.4.2 Effect of Autonomy

For measuring the effect of the autonomy and capabilities of the component databases, the same queries are optimized two different times. The capabilities of the component databases are changed but the relations reside at the same places and the query specifications are the same. The response time and the total time cost are calculated.

The independent variable is the *number of joins* in each query to be optimized. The dependent variable is the *cost* of the optimized queries. Three different configurations of *capabilities of the three component databases* are

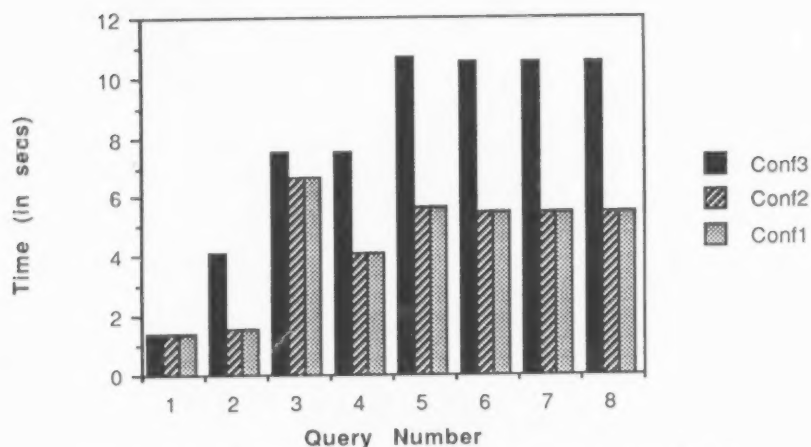


Figure 6.6: Effect of Autonomy (Response Time Cost)

tested, i.e., each database has different capabilities for each configuration. For the first case, the capabilities are those described earlier for the actual multidatabase, i.e., ALTA is the control component database, LSAS is a primary component without temporary storage capabilities and LSDS is a primary component with secondary storage capabilities.

For the second case (Conf2RT and Conf2TT), all the databases have secondary storage capabilities and the control database changes from ALTA to LSDS. For the third case, the ALTA database does not have secondary storage capabilities but LSAS and LSDS do have. LSDS is the control database. The results are shown in Figure 6.6 for response time cost and in Figure 6.7 for total time cost.

It can be observed from the results that when ALTA was made a database

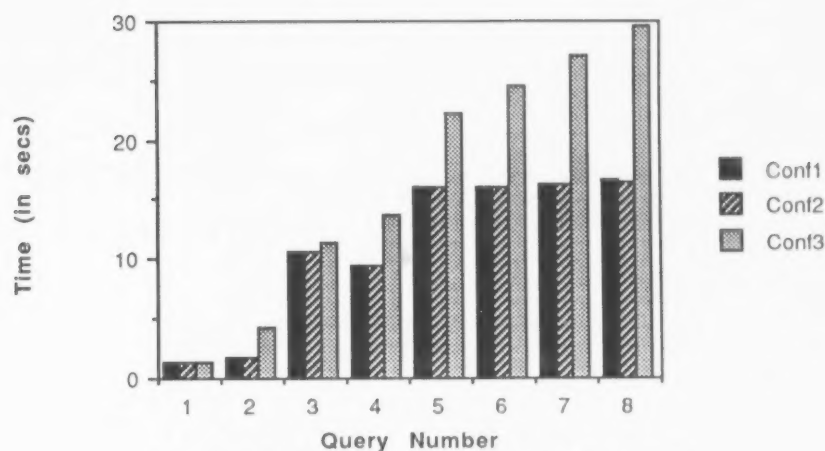


Figure 6.7: Effect of Autonomy (Total Cost)

without secondary storage capabilities, the cost, both response and total cost, increased mainly when the number of joins is high. It can be seen that many joins in the queries require tables from that database and since there is no processing possible there, the rules are moved to a different component. The second case (Conf2RT and Conf2TT) does not show significant changes since only LSAS was added to have secondary storage capabilities and its tables participated in many queries but not as many as ALTA's. For this experiment, the curve is smoother for the three join query. However, there is a high cost for the five join query for response time. The reason for this is similar to that already expressed for the three join query in the previous experiment, i.e., tree not completely bushy because of the number of joins in the query.

6.4.3 Effect of Relation Sizes

To observe the behaviour of the estimated component processing cost, the optimizer varies the size of the participating relations and see how that affects the cost with each term in the cost function as independent variables. The cost is given in response time and total time cost.

The independent variable is the *number of joins*. The dependent variable is the *cost* of the optimized query. The experiment is ran for two different relation sizes or cardinality of relations present in a query. For the first case, Card1, the cardinalities of the relations are taken as shown in Figure 6.3. For the second case, Card2, they were reduced to two thirds of these values. The actual relation cardinalities are shown in Figure 6.3, where only the values for ALTA were estimated as opposed to true values. The results are shown in Figure 6.8 for response time cost and in Figure 6.9 for total time cost.

The difference in the result values is quite small and cannot be observed clearly from the figures. However, the cost was always lower for the second case in which the cardinalities were lower as shown in Figures 6.10 and 6.11. As it can be seen in the cost functions, the processing cost depends on the size of the relations but not in a direct proportional way. The difference in cost may be due more to storage and communication cost than to processing cost. It should be mentioned that for response time cost the delta decreases when the number of joins grows, whereas the opposite happens for total time cost.

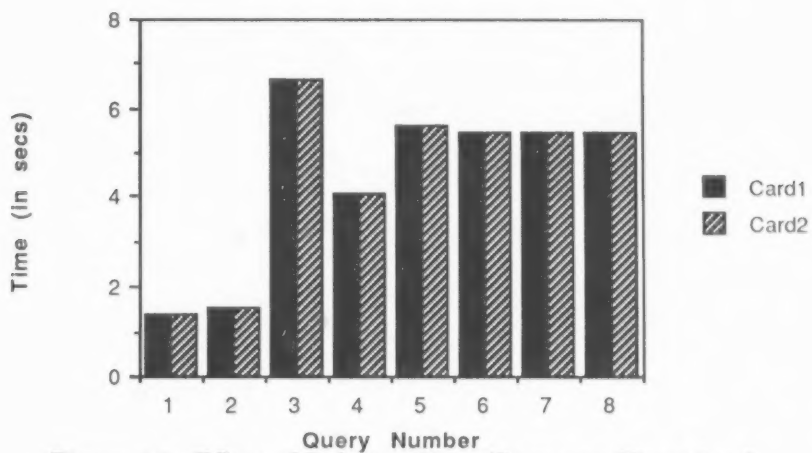


Figure 6.8: Effect of Relation Sizes (Response Time Cost)

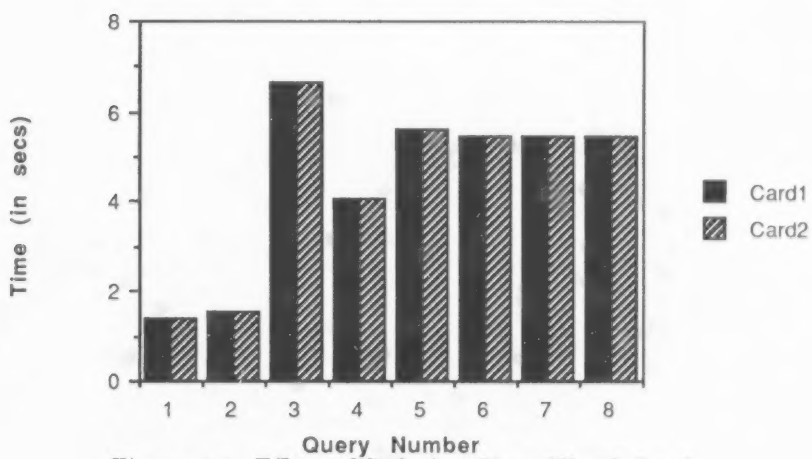


Figure 6.9: Effect of Relation Sizes (Total Cost)

Query Number	Card1RT	Card2RT	Card1RT-Card2RT
1	1.414420	1.410940	0.003480
2	1.533230	1.531940	0.001290
3	6.662290	6.661990	0.000300
4	4.082520	4.082260	0.000260
5	5.643840	5.643620	0.000220
6	5.493760	5.493390	0.000370
7	5.493760	5.493390	0.000370
8	5.493760	5.493390	0.000370

Figure 6.10: Delta Values for Response Time Cost

Query Number	Card1TT	Card2TT	Card1TT-Card2TT
1	1.412120	1.410940	0.0011800
2	1.655350	1.653500	0.0018500
3	10.625500	10.623400	0.0002100
4	9.428660	9.426040	0.0026200
5	15.968900	15.966800	0.0021000
6	16.062099	16.059200	0.0028990
7	16.305300	16.301800	0.0035000
8	16.551399	16.544701	0.0066980

Figure 6.11: Delta Values for Total Time Cost

6.4.4 Effect of Number of Components

To observe how our methodology works depending on the number of components, the relations present in queries with 5 or more joins are mapped to four different number of components. The cost is given in response time and total time cost.

The independent variable is the *number of joins* present in a query. The dependent variable is the *cost* of the optimized query. In the first mapping, relations in the queries are mapped to the three components as shown in the definition of the LRIS system. For the second mapping, relations in the LSAS database are changed to be at LSDS and the LSDS relations are changed to be at LSAS. The results are shown in Figure 6.12 for response time cost and in Figure 6.13 for total time cost.

The results show that better cost was obtained for the second mapping in which there are more relations at the component with secondary storage capabilities than in the component without those capabilities. Therefore, we can see that the distribution of relations among the component databases is important, partly because of the influence of the capabilities of the components.

Another experiment was tried to observe the effect of the number of components. In this experiment, the number of components was varied from one to four. The same queries were optimized for each case. The results are shown in Figure 6.14 for response time cost and in Figure 6.15 for total time cost.

It can be observed from Figure 6.14 that the response time cost of query

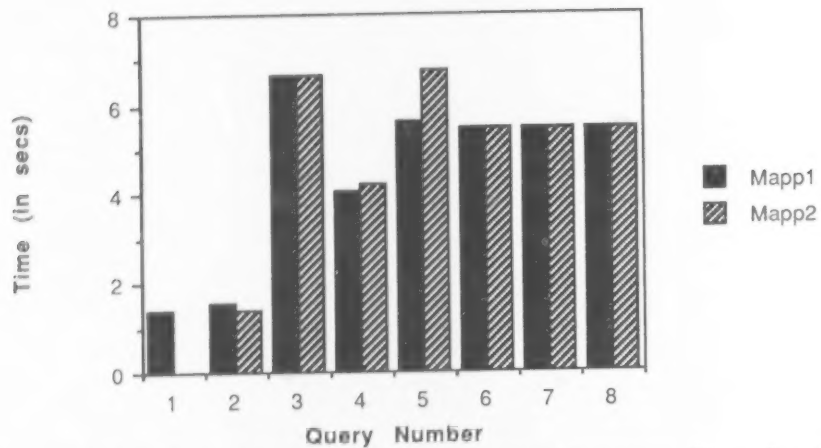


Figure 6.12: Effect of Number of Components (Response Time Cost)

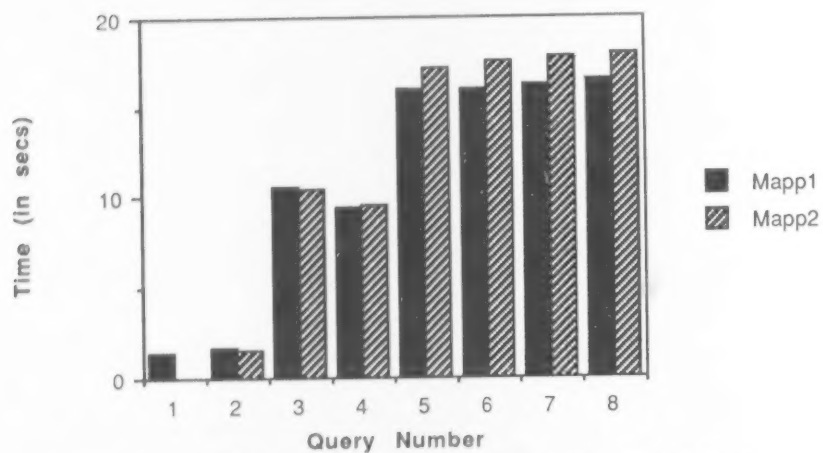


Figure 6.13: Effect of Number of Components (Total Cost)

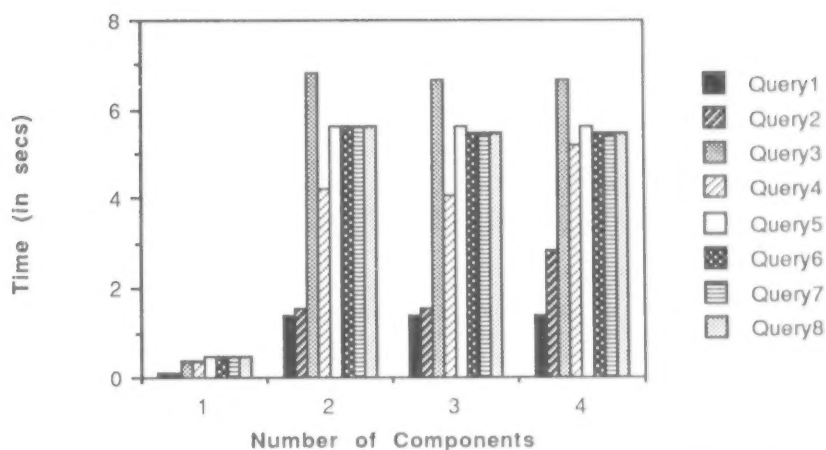


Figure 6.14: Effect of Number of Components (Response Time Cost)

three is still the highest, regardless of the number of components. That is not the case for total time. The other query costs grow with the number of joins. It should also be noted that the cost, both response time and total time, is very low when the component is only one. In that case, there is only processing cost involved. Transmission and temporary storage costs are zero. Once the number of components is increased to two, the curve is very stable for any number of joins in the queries. This is true particularly for response time cost. It still holds for total time cost but there is a small increase in cost for some of the queries when the number of components increases. This may be due to the fact that there is more communication cost involved.

As it can be observed from the data for each experiment, there is a big gap between the total time cost and the response time cost, mainly when the

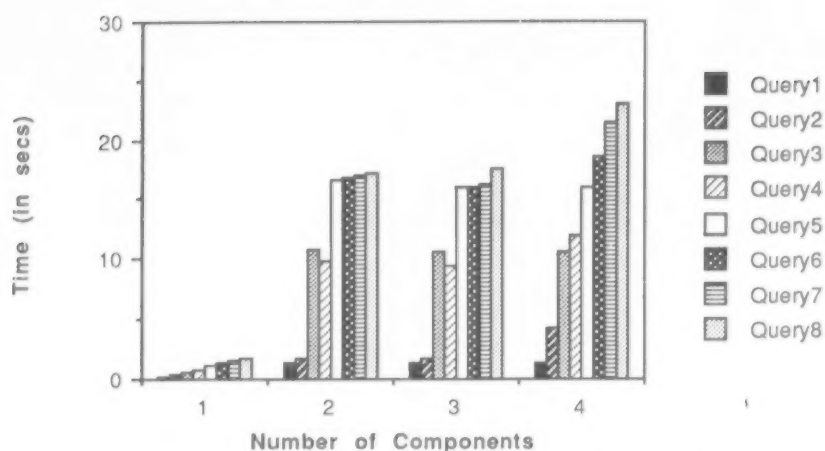


Figure 6.15: Effect of Number of Components (Total Cost)

number of queries increases. This is an important result because it shows that our claim that we exploit parallelism when applying the rules is true. This is due also to the fact that bushy trees are considered in the search space, and the transformation rules applied preserve the search space within bushy trees, leaving the linear trees out of the optimization.

Again, the differences in cost in most of the experiments increase with the number of joins. This also supports our claim that some form of heuristic or randomized technique can be applied as the number of joins in the query increases. Besides, the time to optimize queries which have more than 5 joins is very high due to the combinatorial nature of the search space. The data for this, however, is not shown in this thesis. As the user applications get more complex, the number of joins in queries becomes higher, therefore

it is realistic to think in terms of looking for different alternatives to optimize large queries. An example of this reality is the particular case of our model multidatabase. As it was mentioned earlier, the system has been simplified but the actual number of tables in the component databases is much higher and the system itself is more complex than the one shown for this experimentation.

6.4.5 Comparison of Initial and Optimized Cost

In order to evaluate the performance of the optimizer, the data in Figure 6.16 are presented. The values of initial cost, optimized cost, and their difference are given in seconds. The values correspond to response time cost. The initial cost is the cost of the first tree evaluated when doing optimization. It is not necessarily the highest cost value. Results are shown for total time cost in Figure 6.17. As it can be observed, the initial cost for queries 6, 7 and 8 for response time are the same. This is due to the fact that queries 7 and 8 have trees of the same height as query 6 and just have another branch which is of lower cost. The values change for queries 6, 7 and 8 for total time cost since there is one more join operation.

As it can be seen in the difference column, there is always a difference between the two values in favor of the optimized values. It is zero only for query number 3 in which both values coincide and therefore the difference is zero. It should also be mentioned that the factors (initial transmission cost and transmission factor) used for communication are for very high speed networks, such as ATM networks [BG92]. Therefore, the response time shown

is very short and the difference between the initial value and the optimized one is small. If a more common type of network, such as FDDI network⁴ [Ros86], is tried, the response time values might increase and the difference between the initial and optimized values would also increase and the need for the optimizer would become more obvious.

The optimization time is not shown since the experiments were not run at a stand-alone machine and the performance results may be affected by other processes run at the same machine at the same time. Besides, the optimizer is not a production optimizer and it may be enhanced to give fast optimization time. Also, as it is suggested in Section 7.1 when the number of joins in the query is large a different optimization strategy such as randomized techniques may be used so that optimization time is not extremely long. It should be taken into account the fact that the cost of optimization does not affect the execution cost in the sense that optimization is done at compilation time and not during execution.

⁴FDDI stands for Fiber Distributed Data Interface.

Query No.	Init Cost	Opt Cost	Init - Opt
1	1.414420	1.412120	0.002000
2	1.533340	1.533230	0.000110
3	6.662290	6.662290	0.000000
4	4.086480	4.082520	0.003960
5	5.643850	5.643840	0.000010
6	5.493980	5.493760	0.000220
7	5.493980	5.493760	0.000220
8	5.493980	5.493760	0.000220

Figure 6.16: Initial Tree Cost and Optimized Tree Cost (Response Time Cost)

Query No.	Init Cost	Opt Cost	Init - Opt
1	1.414420	1.412120	0.002000
2	1.655450	1.655350	0.001000
3	10.627400	10.625500	0.001900
4	9.434620	9.428660	0.005660
5	15.975500	15.968900	0.006600
6	16.068400	16.062099	0.006301
7	16.311700	16.305300	0.006400
8	16.555500	16.551399	0.004101

Figure 6.17: Initial Tree Cost and Optimized Tree Cost (Total Time Cost)

Chapter 7

Conclusions and Future Work

7.1 Conclusions

It is imperative that a new approach to query optimization be taken in the case of multidatabase systems. Such an approach must take into account the particular features present in a multidatabase system, mainly those generated by the autonomy and heterogeneity issues. This new trend has been tried out in our approach.

A cost model that embeds component processing cost into the global cost functions is presented, although some of the terms in such functions have to be *estimated* since actual data may not be available from the component databases. Response time cost and total time cost functions are defined.

The search space is described, i.e., the set of valid processing trees is given. To traverse the search space, a set of transformation rules is defined. Rules are defined for the *join* operator for the extended relational algebra that is

presented as part of this work. This algebra includes an operator to show the movement of data among components. In our approach, we consider bushy trees mainly because we want to exploit as much parallelism as possible.

The core part of the optimizer was implemented and the results are shown. The main metric was to compare *total time cost* and *response time cost*. The cost was evaluated considering three types of costs present in multidatabase systems, i.e., component processing cost, communication cost and temporary storage cost. The effect of the transformation rules and heuristics was measured as well. It was shown that the rules that deal with the *move* operator give faster response time costs and total time costs. Therefore, it can be concluded that the use of such rules is appropriate. Regarding the component capabilities, it was found that the cost is affected by it mainly when the number of joins increases. It was found that the cardinality of the participating relations does not affect the cost significantly. This may be due to the way the processing cost is calculated (predicted). In all the experiments, the response time cost was lower than the total time cost. This is due to the fact that we exploit parallelism as much as possible.

7.2 Directions for Future Research

In the presence of large queries, i.e., queries that contain a number of joins higher than 10, randomized techniques may be useful as a search strategy for traversing the search space. Furthermore, heuristics related to the movement of data may be exploited and combined with a randomized technique. Randomized algorithms have also been applied and measured for parallel

execution spaces [LV91], and [LZV93]. Therefore, it may be possible to implement a randomized algorithm to "optimize" the queries, in the presence of many joins. A combination of a randomized algorithm with the appropriate heuristics may lead to getting a good and quick solution to the query even if it is not the "optimal." Because there are cases in which the number of joins is not so large, deterministic optimization is always necessary. Therefore, a parametric type of optimization may be used, i.e., one that recognizes which technique to use depending on the number of joins present.

The implementation of the query processor can be completed. The database may then be queried and the real response time cost may be found. This real value may be compared to the estimated value found by the optimizer.

The future work can be led in at least two different directions. First, the cost functions may become more sophisticated. More parameters may be considered, i.e., variables that may affect the cost of performing a query in a multidatabase environment. Also, methods for calibrating and predicting expected values from the component databases should be developed in order to be able to use accurately estimated values in the computation of the total time and response time cost.

Second, more transformation rules can be developed. More research can be done to find more rules by which the search space is defined. This may lead to a better exploitation of component database capabilities, which will, in turn, lead to finding a good query execution in terms of cost. More features of the search space may be studied in order to compare them to the ones treated in this thesis, i.e., bushy trees.

Bibliography

- [Alb92] Alberta Forestry, Lands and Wildlife, Land Information Services Division. Working paper: A federated schema for the LRIS Network, April 1992.
- [ASD⁺91] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, and M. Shan. The Pegasus multidatabase system. *Computer*, 24(12):19-27, December 1991.
- [BG92] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Inc., 1992.
- [BGMS92] Y. Breitbart, H. GarciaMolina, and A. Silberschatz. Overview of multidatabase transaction management. *The VLDB Journal*, 1(2):181-239, October 1992.
- [BGW⁺81] P. Bernstein, N. Goodman, E. Wong, C. Reeve, and J. Rothnie. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4):602-625, December 1981.

- [BHP92] M. Bright, A. Hurson, and S. Pakzad. A taxonomy and current issues in multidatabase system. *Computer*, 25(3):50-59, March 1992.
- [CBTY89] A. Chen, D. Brill, M. Templeton, and C. Yu. Distributed query processing in a multiple database system. *IEEE Journal on Selected Areas in Communications*, 7(3):390-398, April 1989.
- [CER87] B. Czejdo, D. Embley, and M. Rusinkiewicz. An approach to schema integration and query formulation in federated database systems. In *Proceedings of the Third International Conference on Data Engineering*, pages 477-484, February 1987.
- [Chu90] C. Chung. Dataplex: An access to heterogeneous distributed databases. *Communications of the ACM*, 33(1):70-80, January 1990. (with corrigendum in *Communications ACM*, 4:459).
- [Day83] U. Dayal. Processing queries over generalization hierarchies in a multidatabase system. In *Proceedings of the 9th International Conference on Very Large Databases*, pages 342-353, 1983.
- [DG82] U. Dayal and N. Goodman. Query optimization for codasyl database systems. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 138-150, 1982.
- [DH84] U. Dayal and H. Hwang. View definition and generalization for database integration in MULTIBASE: A system for heteroge-

- neous distributed database. *IEEE Transactions on Software Engineering*, 10(6):628-644, November 1984.
- [DKS92] W. Du, R. Krishnamurthy, and M. Shan. Query optimization in heterogeneous DBMS. In *Proceedings of the 18th International Conference on Very Large Databases*, pages 57-68, August 1992.
- [Du92] W. Du, December 1992. Personal communication.
- [ESW78] R. Epstein, M. Stonebraker, and E. Wong. Distributed query processing in a relational database system. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 169-180, Austin, Texas, May 1978.
- [IC90] Y. Ioannidis and Y. Cha Kang. Randomized algorithms for optimizing large join queries. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 312-321, Atlantic City, NJ, June 1990.
- [IK91] Y. Ioannidis and Y. C. Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 168-177, Denver, Colorado, May 1991.
- [IW87] Y. Ioannidis and E. Wong. Query optimization by simulated annealing. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 9-22, San Francisco, CA, June 1987.

- [KBZ86] R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of nonrecursive queries. In *Proceedings of the 12th International Conference on Very Large Databases*, pages 128-137, Kyoto, Japan, August 1986.
- [LR82] T. Landers and R. L. Rosenberg. An overview of MULTI-BASE. In *Proceedings of the 2nd International Symposium on Distributed Data Bases*, pages 153-184, Berlin, F.R.G., September 1982.
- [LS92] H. Lu and M. Shan. Global query optimization in multidatabase systems. In *Proceeding of the 1992 NFS Workshop on Heterogeneous Databases and Semantic Interoperability*, 1992.
- [LV91] R. Lancelotte and P. Valduriez. Extending the search strategy in a query optimizer. In *Proceedings of the 17th International Conference on Very Large Databases*, pages 363-373, Barcelona, Spain, August 1991.
- [LZV93] R. Lancelotte, M. Zaït, and A. Van Gelder. Measuring the effectiveness of optimization search strategies. *Ingénierie des Systèmes d'Information (ISI)*, 1(2), June 1993.
- [ÖV91] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, Inc., 1991.
- [RC85] M. Rusinkiewicz and B. Czejdo. Query transformation in heterogeneous distributed database systems. In *Proceedings of 5th In-*

ternational Conference on Distributed Computing Systems, pages 300-307, Denver, Colorado, May 1985.

- [RC87] M. Rusinkiewicz and B. Czejdo. An approach to query processing in federated database systems. In *Proceedings of the 20th Hawaii International Conference on System Sciences*, pages 430-440, Kailua-Kona, Hawaii, January 1987.
- [RC89] M. Rusinkiewicz and B. Czejdo. Query transformation in a multi-database environment using a universal symbolic manipulation system. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 46-53, June 1989.
- [Ros86] F. Ross. Fddi-a Tutorial. *IEEE Communications*, 24(5):10-17, May 1986.
- [SG88] A. Swami and A. Gupta. Optimization of large join queries. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 8-17, June 1988.
- [SL90] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183-236, September 1990.
- [Swa89] A. Swami. Optimization of large join queries: Combining heuristics and combinatorial techniques. In *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pages 367-376, June 1989.

- [TBD⁺87] M. Templeton, D. Brill, S. Dao, E. Lund, P. Ward, L. P. Chen, and R. MacGregor. Mermaid: A front-end to distributed heterogeneous databases. In *Proceedings of the IEEE*, pages 695–707, May 1987.
- [TTC⁺90] G. Thomas, G. Thompson, C. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, and B. Hartman. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22(3):237–266, September 1990.
- [Ull82] J. Ullman. *Principles of Database Systems*. Rockville, Md.: Computer Science Press, 1982.
- [Zhu92] Q. Zhu. Query optimization in multidatabase systems. *Proceedings of the 1992 CASCON Conference*, 2:111–127, November 1992.

Appendix A

Table Descriptions and Attributes

The tables in the ALTA database are the following:

- Title. This table contains information about the titles of a piece of land, i.e., about the ownership of the land.
- Parcel. This table contains information about parcels or pieces of land that are identified by the LINC number.
- Instrument. This table contains information about the possible instruments on a piece of land.
- Owner. This table contains information about the parties or owners of properties.
- Owner/Title. This table establishes the relationship between the titles of pieces of land and their parties or owners .

The tables in the LSAS database are the following:

- Client (CLI). This table contains information about *clients* who perform certain *activities* on a piece of land.
- ATS Land. This table contains information about the localization of a piece of land and details about its characteristics.
- ATS Land/CLIENT. This table contains information about the clients who perform activities on a piece of land.
- Activity. This table contains the description of the activities that may be developed at any piece of land.

The tables in the LSDS database are the following:

- Municipality. This table contains the code and description of each municipality in the province.
- Plan. This table contains information about the plan number on a piece of land.
- ATS Land/Municipality. This table relates a piece of land with its municipality.

Title		
Title Reference Number	Char	12
LINC Number	Char	10
Plan Registration Number	Char	7
Municipality Land Desc	Char	60
Registration Date	Char	19
Title Rights Type	Char	1

Parcel (PAR)		
LINC Number	Char	10
Short Legal	Char	40
Area By LINC Number Text	Char	63
Condominium Shares Text	Char	80

Owner (OWN)		
Owner Id	Char	5
Owner Name	Char	80
Owner Occupation	Char	60
Owner Address 1	Char	120
Owner Address 2	Char	120
Owner Address 3	Char	120
Owner Address 4	Char	120
Owner Province	Char	25
Owner Postal Code	Char	10

Instrument (INS)		
Instrument Id	Char	20
Title Reference Number	Char	12
Instrument Type	Char	4
Instrument Type Text	Char	60
Instrument Text	Char	80
Lease Commencement Date	Char	19
Lease Term Date	Char	19
Lease Text	Char	60

Owner/Title (OWN_TITLE)		
Owner Id	Char	5
Title Reference Number	Char	12

Figure A.1: ALTA Tables

CLIENT (CLI)

Client ID	Char	7
Address Number	Char	3
Client Name	Char	70
Client Name 2	Char	35
Address	Char	35
City	Char	35
Province	Char	24
Country	Char	20
Postal Code	Char	10

ATS Land (ATS)

ATS Land ID	Char	20
Administrative Status	Char	10
Parcel Area	Char	6
Water Code	Char	4

ATS Land/CLIENT (ATS_CLI)

ATS Land ID	Char	20
Client ID	Char	7
Activity	Char	16

Activity (ACT)

Activity	Char	16
Description	Char	60

Figure A.2: LSAS Tables

ATS Land/Municipality (ATS_Munic)

ATS Land ID	Char	20
Municipality Code	Char	4

Municipality (MUNIC)

Municipality Code	Char	4
Description	Char	60

Plan (PLAN)

Plan Registration Number	Char	7
ATS Land ID	Char	20
Registration Date	Char	19

Figure A.3: LDS Tables